Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 1: Preface

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

About This Programmer's Reference Manual 1

Organization of the PRM 7

Device Tags and Definitions 8

Reserved Bits and Software Compatibility 9

Terminology 10

About This Programmer's Reference Manual

This Preface serves as an Introduction to the Programmer's Reference Manual (PRM) for the Graphics Processing Unit (GPU) included on the Kaby Lake series of chips. The PRM includes both narrative content that explains conceptually how the different modules of the GPU work, and a comprehensive Command Reference that describes named memory addresses/registers, commands, structures and enumerations, at the bit-level.

A Public View of the PRM

For each major chip series, Intel publicly releases a version of the PRM for the Open Source development community and developers at large. For those releases, we release the PRM in Portable Document Format (PDF).

In Multiple PDF Files

Due to file size constraints in generating PDF output (the Kaby Lake "Open Source" version of the PRM is roughly 5,000 pages long), the PRM is generated in multiple PDF files. The organization of those files reflects the major components of the Kaby Lake GPU. Both the narrative content and the Command Reference had to be partitioned into multiple PDF files.

Without Cross-book Links

Cross-book hyperlinks could not reasonably be included for the PDF. Instead, those links have been disabled and colored dark red and display in a serif font for easy recognition. This is an example:

RING_BUFFER_TAIL – Ring Buffer Tail

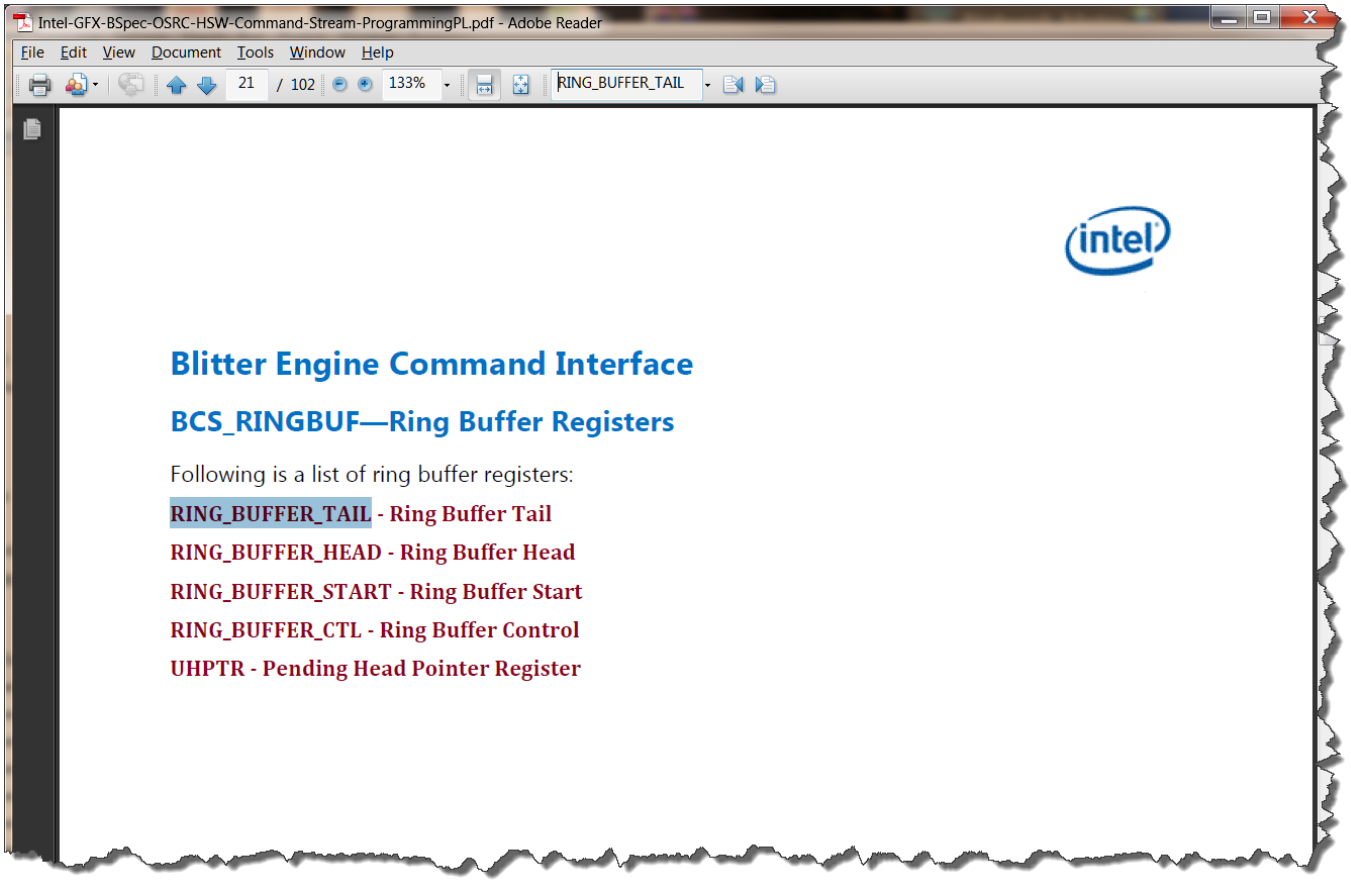
RING_BUFFER_HEAD – Ring Buffer Head

Searching Across Multiple PDF Files

You can quickly access the detailed explanation for each named memory address, command, structure or enumerator utilizing the Advanced Search capabilities in Acrobat Reader.

To maximize the efficiency of the search, make sure to place all PRM PDFs in a single folder, or at least the four Command Reference PDFs in a single folder.

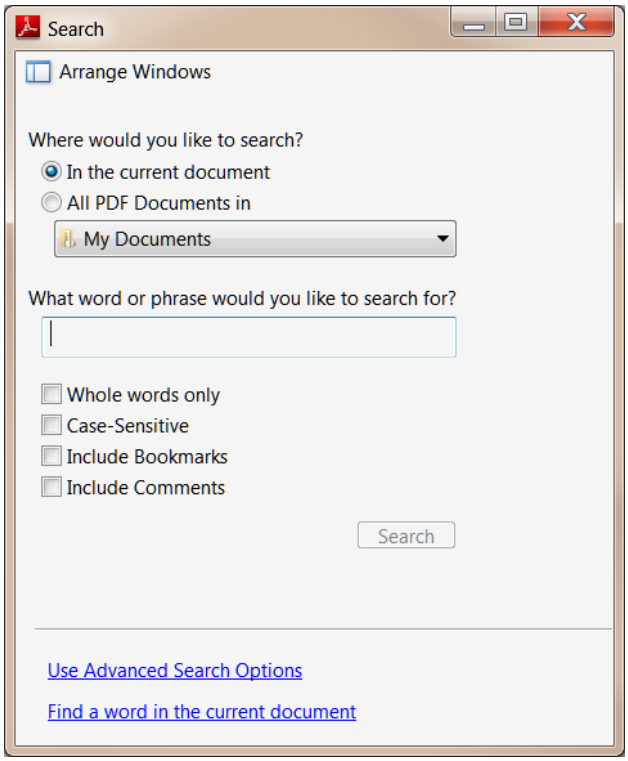
Let's walk through an example PDF search … you might be reading the Command Stream Programming module PDF file and encounter the page illustrated below.



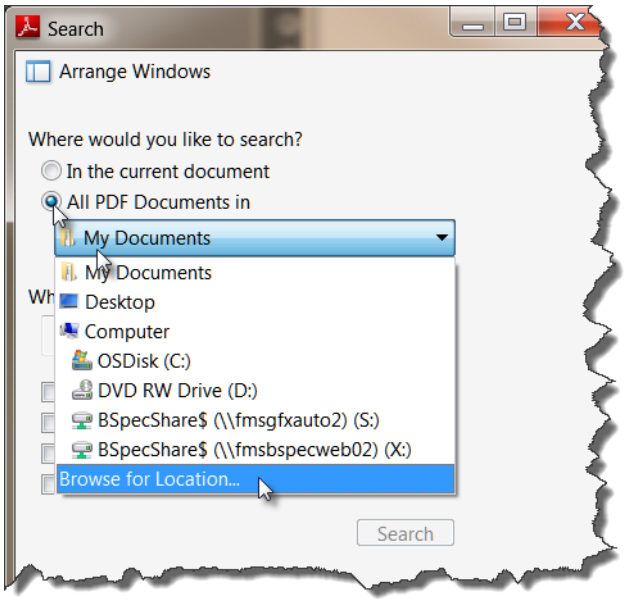
To view the detailed explanation of one of the listed registers … for this example, we will use **RING_BUFFER_TAIL …**

1. Simply select **Search** or **Advanced Search** from the **Edit** menu, depending on the version of Acrobat you might be using. (If you are using some other PDF Reader, it probably will also have advanced search capabilities.)

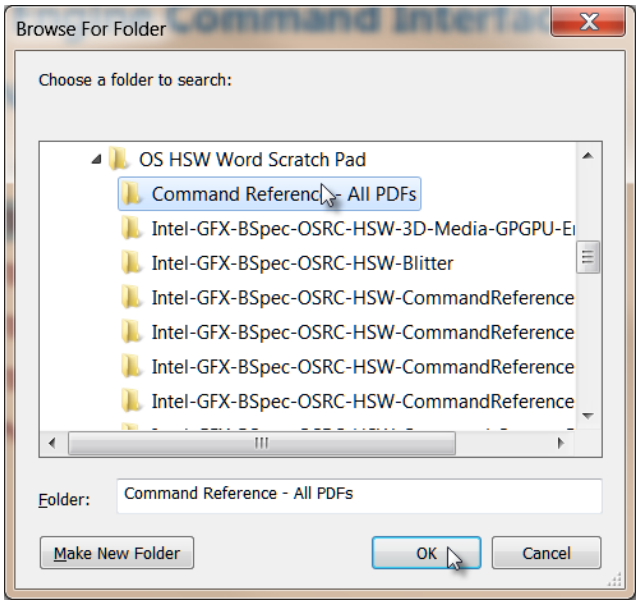
A search dialog box then displays.



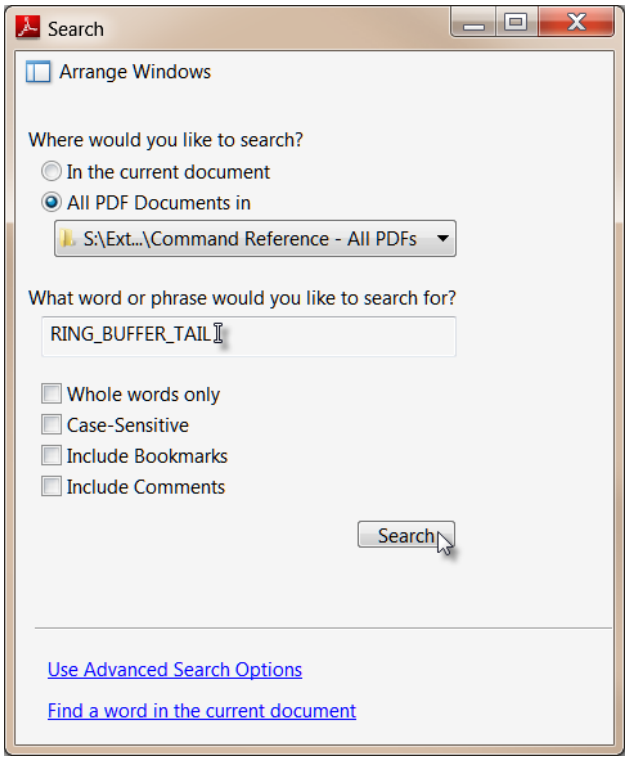
2. Select the **All PDF Documents in** radio button, click the pull-down list box, then select **Browse for Location...**



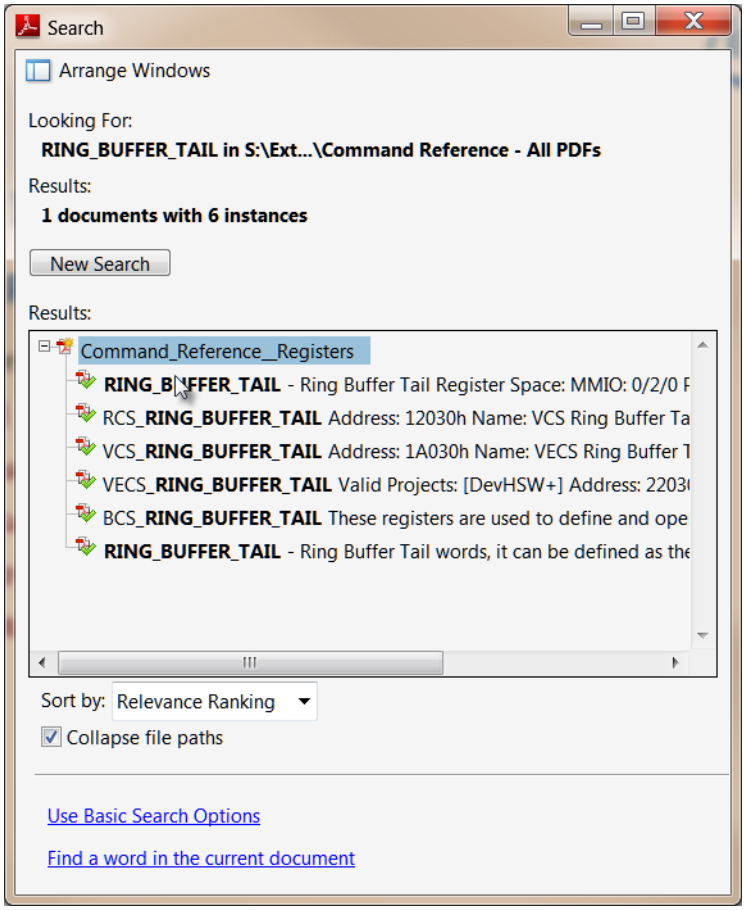
3. Browse to and select the folder containing the pertinent PDF files and click **OK**.



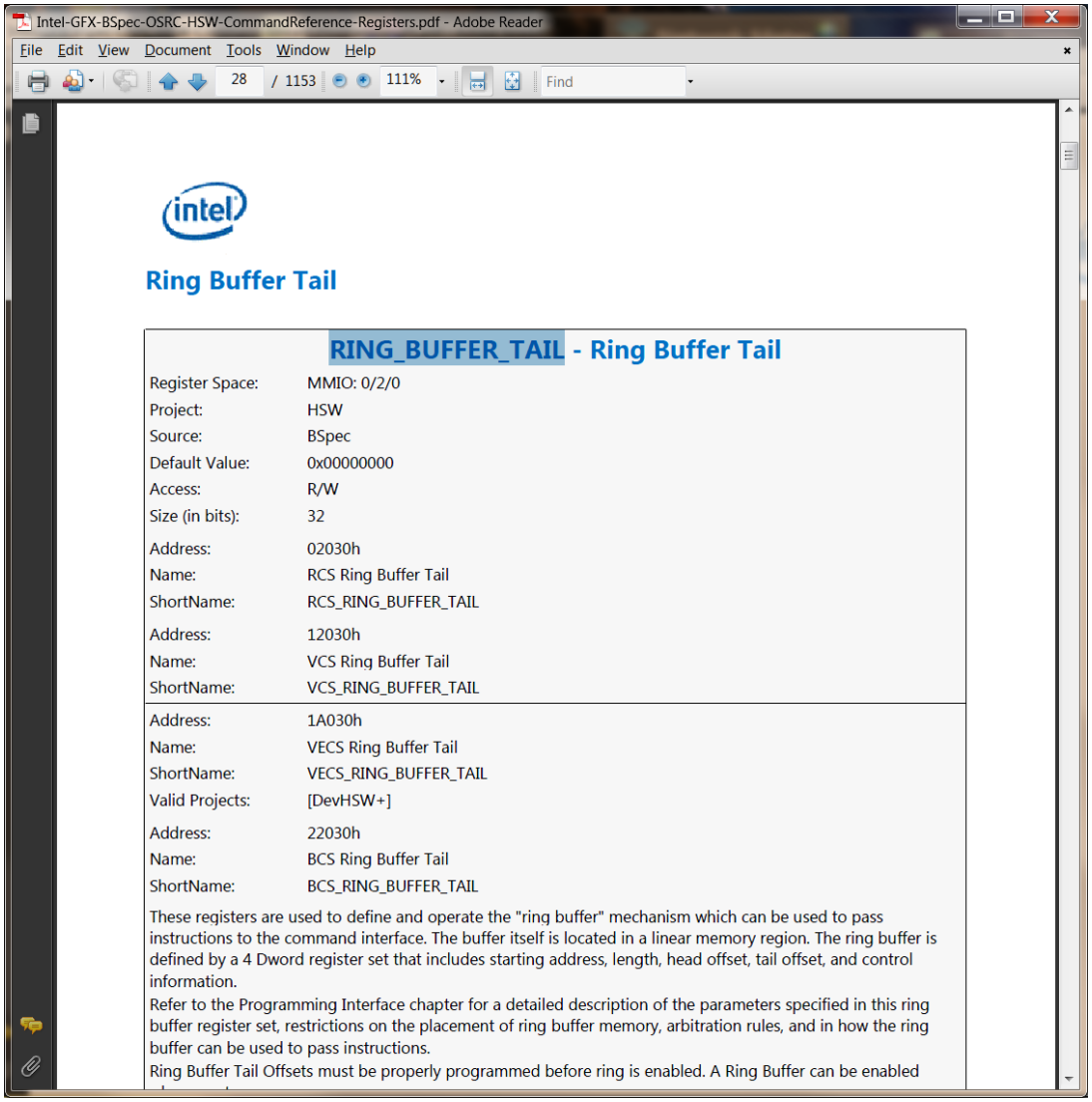
4. The Browse for Folder dialog closes and the focus shifts back to the original Search dialog. Enter the word or phrase from the red, serif text in the **What word or phrase would you like to search for?** text box and click **Search**.



As matches are found, they will be listed. When the search is complete, the dialog will appear similar to the following:



Click the most relevant result and the PDF file containing that result will open at the correct page.



Organization of the PRM

The Intel® Graphics PRM is organized into the following content areas:

Command Reference provides a tabular graphical interface tool for searching for and displaying command opcodes, registers, structures, and instructions.

GPU Overview introduces the GPU and its subsystems. This includes high level descriptions of its hardware pipelines, command formats, register maps, and supported memory/surface formats.

Configurations tracks the evolution of the graphics architecture and provides generational information for many basic architectural attributes.

Memory Views covers the Graphics Memory Interface, including memory interface functions, tiling, the physical graphics memory, page faults and error handling, memory types, common video and non-video surface formats, and other information related to the memory interface.

Command Stream Programming describes the programming interfaces for Command Streamer, which manages the use of the 3D and Media pipelines. It performs switching between pipelines and forwarding command streams to the currently active pipeline. It also manages the allocation of the Unified Return Buffer (URB) and helps support the Constant URB Entry (CURBE) function. It includes the following topics:

Topic
Command Formats
Blitter Engine Command Interface
Render Engine Command Interface
Video Codec Engine Command Interface
Video Enhancement Engine Command Interface
Preemption
Command Streamer (CS) ALU Programming
Resource Streamer

3D/Media/GPGPU Engine includes the following topics:

- Command stream backend processing
- The Graphics L3 large-storage cache, which also acts as a GFX Unified Return Buffer (URB)
- Shared Local Memory (SLM)
- Messaging
- Shared functions for the 3D pipeline
- 3D fixed functions
- Media/GPGPU Pipeline processing via both fixed functions and programmable GPU cores
- Execution units

Media VDBOX covers the **Multi-Format Codec (MFX) Engine**, the hardware fixed-function pipeline that includes both multi-format decoding (MFD) and multi-format encoding (MFC).

Media VEBOX discusses the Video Enhancement pipeline, an independent pipe that manages image enhancement functions. Topics include denoise filtering, deinterlacing, color processing, auto contrast enhancement, the capture pipe, output statistics, and video enhancement engine functions.

HEVC covers the High Efficiency Video Coding (HEVC) Codec Pipeline (HCP), which is a fixed function hardware video codec pipeline responsible for encoding and decoding HEVC streams, with a target resolution of 4kx2k at 60 frames per second.

Blitter discusses the **BL**ock **I**mage **T**ransfer**er** Engine for 2D graphics processing.

Display describes display engine registers, including display HD audio configuration and programming. It includes VGA and extended VGA registers, CPU display registers, North display engine registers, and South display engine registers. The Display section explains audio codec verbs, display audio configuration registers, and the display audio controller registers.

Memory Mapped I/O (**MMIO**) includes information on Slice Registers and Die Recovery Units including Start and End Range offsets.

SFC discusses the commands sent to the **Scalar and Format Converter (SFC)** pipeline from the Media VEBOX and Media VDBOX.

Observability Performance Counters discusses performance visibility, including performance event

Workarounds provide software workarounds for known issues with the GPU.

Device Tags and Definitions

The following table lists device "tags" (abbreviations, projects) used in various parts of this document as aliases for the device names. PRM content without any device tagging applies to all devices in the current spec.

Program Name	Graphics Architecture	Description
Kaby Lake	Gen9p5	An update to the Skylake microarchitecture, using a 14nm process.

Reserved Bits and Software Compatibility

In many register, instruction, and memory layout descriptions, certain bits are marked as “Reserved.” When bits are marked as reserved, it is essential for compatibility with future devices that the software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as undefined *and unpredictable*. Software should follow the guidelines listed below in dealing with reserved bits:

1. Do not depend on the states of any reserved bits when testing values of registers that contain such bits.
2. Mask out the reserved bits before testing.
3. Do not depend on the states of any reserved bits when storing to an instruction or to a register.
4. When loading a register or formatting an instruction, always load the reserved bits with the values indicated in the documentation (if any), or reload them with the values previously read from the register.

Terminology

Term	Abbr.	Definition
3D Pipeline	--	One of the two pipelines supported in the GPE. The 3D Pipeline is a set of fixed-function units arranged in a pipelined fashion, which process 3D-related commands by spawning EU threads. Typically this processing includes rendering primitives. See *3D Pipeline*.
Adjacency	--	One can consider a single line object as existing in a strip of connected lines. The neighboring line objects are called "adjacent objects", with the non-shared endpoints called the "adjacent vertices." The same concept can be applied to a single triangle object, considering it as existing in a mesh of connected triangles. Each triangle shares edges with three other adjacent triangles, each defined by a non-shared adjacent vertex. Knowledge of these adjacent objects/vertices is required by some object processing algorithms (e.g., silhouette edge detection). See *3D Pipeline*.
Application IP	AIP	Application Instruction Pointer. This is part of the control registers for exception handling for a thread. Upon an exception, hardware moves the current IP into this register and then jumps to System Instruction Pointer (SIP).
Architectural Register File	ARF	A collection of architecturally visible registers for a thread such as address registers, accumulator, flags, notification registers, IP, null, etc.
Binding Table	--	Memory-resident list of pointers to surface state blocks (also in memory).
Binding Table Pointer	BTP	Pointer to a binding table, specified as an offset from the Surface State Base Address register.
Blitter	BLT	Block Image Transferrer
Bypass Mode	--	Mode where a given fixed function (FF) unit is disabled and forwards data down the pipeline unchanged. Not supported by all FF units.
Byte	B	A numerical data type of 8 bits, B represents a signed byte integer.
Child Thread	--	A branch-node or a leaf-node thread that is created by another thread. It is a kind of thread associated with the media fixed function pipeline. A child thread is originated from a thread (the parent) executing on an EU and forwarded to the Thread Dispatcher by the TS unit. A child thread may or may not have child threads depending on whether it is a branch-node or a leaf-node thread. All pre-allocated resources such as URB and scratch memory for a child thread are managed by its parent thread. See also *Parent Thread*.
Clip Space	--	A 4-dimensional coordinate system within which a clipping frustum is defined. Object positions are projected from Clip Space to NDC space via "perspecitive divide" by the W coordinate, and then viewport mapped into Screen Space.
Clipper	--	3D fixed function unit that removes invisible portions of the drawing sequence by discarding (culling) primitives or by "replacing" primitives with one or more primitives that replicate only the visible portion of the original primitive.
Color Calculator	CC	Part of the Data Port shared function, the color calculator performs fixed-function pixel operations (e.g., blending) before writing a result pixel into the render cache.

Term	Abbr.	Definition
Command	--	Directive fetched from a ring buffer in memory by the Command Streamer and routed down a pipeline. Should not be confused with instructions which are fetched by the instruction cache subsystem and executed on an EU.
Command Streamer	CS or CSI	Functional unit of the Graphics Processing Engine that fetches commands, parses them, and routes them to the appropriate pipeline.
Constant URB Entry	CURBE	A UE that contains "constant" data for use by various stages of the pipeline.
Control Register	CR	The read-write registers are used for thread mode control and exception handling for a thread.
Core	--	Alternative name for an EU in the multi-processor system. See *EU*.
Data Port	DP	Shared function unit that performs a majority of the memory access types on behalf of SNB+ programs. The Data Port contains the render cache and the constant cache and performs all memory accesses requested by SNB+ programs except those performed by the Sampler. See DataPort.
Degenerate Object	--	Object that is invisible due to coincident vertices or because it does not intersect any sample points (usually due to being tiny or a very thin sliver).
Destination	--	Describes an output or write operand.
Destination Size	--	The number of data elements in the destination of a SIMD instruction.
Destination Width	--	The size of each of (possibly) many elements of the destination of a SIMD instruction.
Double Quad word (DQword)	DQ	A fundamental data type, DQ represents 16 bytes.
Double word (DWord)	D or DW	A fundamental data type, D or DW represents 4 bytes.
Drawing Rectangle	--	A screen-space rectangle within which 3D primitives are rendered. An object's screen-space positions are relative to the Drawing Rectangle origin. See *Strips and Fans*.
End of Block	EOB	A 1-bit flag in the non-zero DCT coefficient data structure indicating the end of an 8x8 block in a DCT coefficient data buffer.
End Of Thread	EOT	A message sideband signal on the Output message bus signifying that the message requester thread is terminated. A thread must have at least one SEND instruction with the EOT bit in the message descriptor field set to properly terminate.
Exception	--	Type of (normally rare) interruption to EU execution of a thread's instructions. An exception occurrence causes the EU thread to begin executing the System Routine, which is designed to handle exceptions.
Execution Channel	--	Single lane of a SIMD operand.
Execution Size	ExecSize	Execution Size indicates the number of data elements processed by an SIMD instruction. It is one of the instruction fields and can be changed per instruction.
Execution Unit	EU	An EU is a multi-threaded processor within the multi-processor system. Each EU is a fully-capable processor containing instruction fetch and decode, register files, source operand swizzle and SIMD ALU, etc. An EU is also referred to as a core.

Term	Abbr.	Definition
Execution Unit Identifier	EUID	The 4-bit field within a thread state register (SR0) that identifies the row and column location of the EU where a thread is located. A thread can be uniquely identified by the EUID and TID.
Execution Width	ExecWidth	The width of each of several data elements that may be processed by a single SIMD instruction.
Extended Math Unit	EM	A Shared Function that performs more complex math operations on behalf of several EUs.
Fixed Function	FF	Function of the pipeline that is performed by dedicated (vs. programmable) hardware.
Fixed Function ID	FFID	Unique identifier for a fixed function unit.
Gateway	GW	See *Message Gateway*.
General Register File	GRF	Large read/write register file shared by all the EUs for operand sources and destinations. This is the most commonly used read-write register space organized as an array of 256-bit registers for a thread.
General State Base Address	--	The Graphics Address of a block of memory-resident “state data”, which includes state blocks, scratch space, constant buffers, and kernel programs. The contents of this memory block are referenced via offsets from the contents of the General State Base Address register. See *Graphics Processing Engine*.
Geometry Shader	GS	Fixed-function unit between the vertex shader and the clipper that (if enabled) dispatches “geometry shader” threads on its input primitives. Application-supplied geometry shaders normally expand each input primitive into several output primitives to perform 3D modeling algorithms such as fur/fins.
Graphics Address	--	The GPE virtual address of some memory-resident object. This virtual address gets mapped by a GTT or PGTT to a physical memory address. Note that many memory-resident objects are referenced not with Graphics Addresses, but instead with offsets from a “base address register”.
Graphics Processing Engine	GPE	Collective name for the Subsystem, the 3D and Media pipelines, and the Command Streamer.
Guardband	GB	Region that may be clipped against to make sure objects do not exceed the limitations of the renderer's coordinate space.
Horizontal Stride	HorzStride	The distance in element-sized units between adjacent elements of a region-based GRF access.
Immediate floating point vector	VF	A numerical data type of 32 bits, an immediate floating point vector of type VF contains 4 floating point elements with 8 bits each. The 8-bit floating point element contains a sign field, a 3-bit exponent field and a 4-bit mantissa field. It may be used to specify the type of an immediate operand in an instruction.
Immediate integer vector	V	A numerical data type of 32 bits, an immediate integer vector of type V contains 8 signed integer elements with 4 bits each. The 4-bit integer element is in 2’s complement form. It may be used to specify the type of an immediate operand in an instruction.
Index Buffer	IB	Buffer in memory containing vertex indices.

Term	Abbr.	Definition
Intel Architecture	IA	
Instance	--	In the context of the VF unit, an instance is one of a sequence of sets of similar primitive data. Each set has identical vertex data but may have unique instance data that differentiates it from other sets in the sequence.
Instruction	--	Data in memory directing an EU operation. Instructions are fetched from memory, stored in a cache, and executed on one or more cores. Not to be confused with commands which are fetched and parsed by the command streamer and dispatched down the 3D or Media pipeline.
Instruction Pointer	IP	The address (really an offset) of the instruction currently being fetched by an EU. Each EU has its own IP.
Instruction Set Architecture	ISA	The ISA describes the instructions supported by an EU.
Instruction State Cache	ISC	On-chip memory that holds recently-used instructions and state variable values.
Interface Descriptor	--	Media analog of a State Descriptor.
Intermediate Z	IZ	Completion of the Z (depth) test at the front end of the Windower/Masker unit when certain conditions are met (no alpha, no pixel-shader computed Z values, etc.).
Inverse Discrete Cosine Transform	IDCT	The stage in the video decoding pipe between IQ and MC.
Inverse Quantization	IQ	A stage in the video decoding pipe between IS and IDCT.
JIT	--	Just-in-time compiler, aka "jitter".
Kernel	--	A sequence of instructions that is logically part of the driver or generated by the jitter. Differentiated from a Shader which is an application supplied program that is translated by the jitter to instructions.
Logical Ring Context Area	LRCA	Memory area used to store contents of registers and state information required for initiating and resuming communication between software application and hardware graphics pipeline via Ring Buffers.
Least Significant Bit	LSB	The bit with the lowest bit position within a group of bits, which could be a bit group, DWord, field, instruction, memory range, register, or structure. For example, bit 0 of a DWord.
MathBox	--	See *Extended Math Unit*
Media	--	Term for operations such as video decode and encode that are normally performed by the Media pipeline.
Media Pipeline	--	Fixed function stages dedicated to media and “generic” processing, sometimes referred to as the generic pipeline.
Memory-mapped Input/Output	MMIO	A methord for performing input/output between the CPU/GPU and peripheral devices.
Message	--	Messages are data packages transmitted from a thread to another thread, another shared function, or another fixed function. Message passing is the primary communication mechanism of the architecture.

Term	Abbr.	Definition
Message Gateway	--	Shared function that enables thread-to-thread message communication/synchronization, used solely by the Media pipeline.
Most Significant Bit	MSB	The bit with the highest bit position within a group of bits, which could be a bit group, DWord, field, instruction, memory range, register, or structure. For example, bit 31 of a DWord.
Motion Compensation	MC	Part of the video decoding pipe.
Motion Picture Expert Group	MPEG	MPEG is the international standard body JTC1/SC29/WG11 under ISO/IEC that has defined audio and video compression standards such as MPEG-1, MPEG-2, and MPEG-4.
Motion Vector Field Selection	MVFS	A four-bit field selecting reference fields for the motion vectors of the current macroblock.
Multiple Render Targets	MRT	Multiple independent surfaces that may be the target of a sequence of 3D or Media commands that use the same surface state.
Normalized Device Coordinates	NDC	Clip Space Coordinates that have been divided by the Clip Space “W” component.
Object	--	A single triangle, line, or point.
OpenGL	OGL	**O**pen **G**raphics **L**ibrary. A graphics API specification associated with Linux.
Parent Thread	--	A thread corresponding to a root-node or a branch-node in thread generation hierarchy. A parent thread may be a root thread or a child thread depending on its position in the thread generation hierarchy.
Pipeline Stage	--	An abstracted element of the 3D Pipeline, providing functions performed by a combination of the corresponding hardware FF unit and the threads spawned by that FF unit.
Pipelined State Pointers	PSP	Pointers to state blocks in memory that are passed down the pipeline.
Pixel Shader	PS	Shader that is supplied by the application, translated by the jitter and is dispatched to the EU by the Windower (conceptually) once per pixel.
Point	--	A drawing object characterized only by position coordinates and width.
Primitive	--	Synonym for object: triangle, rectangle, line, or point.
Primitive Topology	--	A composite primitive such as a triangle strip or a line list. Also includes the objects triangle, line, and point as degenerate cases.
Provoking Vertex	--	The vertex of a primitive topology from which vertex attributes that are constant across the primitive are taken.
Quad Quad word (QQword)	QQ	A fundamental data type, QQ represents 32 bytes.
Quad Word (QWord)	QW	A fundamental data type, QW represents 8 bytes.
Rasterization	--	Conversion of an object represented by vertices into the set of pixels that make up the object.

Term	Abbr.	Definition
Region-based addressing	--	Collective term for the register addressing modes available in the EU instruction set that permit discontiguous register data to be fetched and used as a single operand.
Render Cache	RC	Cache in which pixel color and depth information is written before being written to memory, and where prior pixel destination attributes are read in preparation for blending and Z test.
Render Target	RT	A destination surface in memory where render results are written.
Render Target Array Index	--	Selector of which of several render targets the current operation is targeting.
Resource Streamer	RS	Functional unit of the Graphics Processing Engine that examines the commands in the ring buffer in an attempt to pre-process certain long latency items for the remainder of the graphics processing.
Root Thread	--	A root-node thread. A thread corresponds to a root-node in a thread generation hierarchy. It is a kind of thread associated with the media fixed function pipeline. A root thread is originated from the VFE unit and forwarded to the Thread Dispatcher by the TS unit. A root thread may or may not have child threads. A root thread may have scratch memory managed by TS. A root thread with children has its URB resource managed by the VFE.
Sampler	--	Shared function that samples textures and reads data from buffers on behalf of EU programs.
Scratch Space	--	Memory allocated to the subsystem that is used by EU threads for data storage that exceeds their register allocation, persistent storage, storage of mask stack entries beyond the first 16, etc.
Shader	--	A program supplied by the application in a high level shader language, and translated to instructions by the jitter.
Shared Function	SF	Function unit that is shared by EUs. EUs send messages to shared functions, that consume the data and may return results. The Sampler, Data Port, and Extended Math unit are all shared functions.
Shared Function ID	SFID	Unique identifier used by kernels and shaders to target shared functions and to identify their returned messages.
Single Instruction Multiple Data	SIMD	A parallel processing architecture that exploits data parallelism at the instruction level. It can also be used to describe the instructions in such an architecture or to describe the amount of data parallelism in a particular instruction (SIMD8 for example).
Source	--	Describes an input or read operand.
Spawn	--	To initiate a thread for execution on an EU. Done by the thread spawner as well as most FF units in the 3D Pipeline.
Sprite Point	--	Point object using full range texture coordinates. Points that are not sprite points use the texture coordinates of the point's center across the entire point object.
State Descriptor	--	Blocks in memory that describe the state associated with a particular FF, including its associated kernel pointer, kernel resource allowances, and a pointer to its surface state.

Term	Abbr.	Definition
State Register	SR	The read-only registers containing the state information of the current thread, including the EUID/TID, Dispatcher Mask, and System IP.
State Variable	SV	An individual state element that can be varied to change the way given primitives are rendered or media objects processed. On state variables persist only in memory and are cached as needed by rendering/processing operations except for a small amount of non-pipelined state.
Stream Output	--	A term for writing the output of a FF unit directly to a memory buffer instead of, or in addition to, the output passing to the next FF unit in the pipeline. Currently only supported for the Geometry Shader (GS) FF unit.
Strips and Fans	SF	Fixed function unit whose main function is to decompose primitive topologies such as strips and fans into primitives or objects.
Sub-Register	--	Subfield of a SIMD register. A SIMD register is an aligned fixed size register for a register file or a register type. For example, a GRF register, *r2*, is a 256-bits wide, 256-bit aligned register. A sub-register, *r2.3:d*, is the fourth dword of GRF register *r2*.
Subsystem	--	The name given to the resources shared by the FF units, including shared functions and EUs.
Surface	--	A rendering operand or destination, including textures, buffers, and render targets.
Surface State Base Pointer	--	Base address used when referencing binding table and surface state data.
Synchronized Root Thread	--	A root thread that is dispatched by TS upon a 'dispatch root thread' message.
System IP	SIP	There is one global System IP register for all the threads. From a thread's point of view, this is a virtual read only register. Upon an exception, hardware performs some bookkeeping and then jumps to SIP.
System Routine	--	Sequence of instructions that handles exceptions. SIP is programmed to point to this routine, and all threads encountering an exception will call it.
Thread	--	An instance of a kernel program executed on an EU. The life cycle for a thread starts from the executing the first instruction after being dispatched from Thread Dispatcher to an EU to the execution of the last instruction – a *send* instruction with EOT that signals the thread termination. Threads in the system may be independent from each other or communicate with each other through Message Gateway share function.
Thread Dispatcher	TD	Functional unit that arbitrates thread initiation requests from Fixed Functions units and instantiates the threads on EUs.
Thread Identifier	TID	The field within a thread state register (SR0) that identifies which thread slots on an EU a thread occupies. A thread can be uniquely identified by the EUID and TID.
Thread Payload	--	Before a thread starting execution, some amount of data is pre-loaded into the thread's GRF (starting at r0). This data is typically a combination of control information provided by the spawning entity (FF Unit) and data read from the URB.

Term	Abbr.	Definition
Thread Spawner	TS	The second and the last fixed function stage of the media pipeline that initiates new threads on behalf of generic/media processing.
Topology	--	See *Primitive Topology*.
Unified Return Buffer	URB	The on-chip memory managed/shared by Fixed Functions in order for a thread to return data that will be consumed either by a Fixed Function or other threads.
Unsigned Byte integer	UB	A numerical data type of 8 bits.
Unsigned Double Word integer	UD	A numerical data type of 32 bits. It may be used to specify the type of an operand in an instruction.
Unsigned Word integer	UW	A numerical data type of 16 bits. It may be used to specify the type of an operand in an instruction.
Unsynchronized Root Thread	--	A root thread that is automatically dispatched by TS.
URB Dereference	--	
URB Entry	UE	URB Entry: A logical entity stored in the URB (such as a vertex), referenced via a URB Handle.
URB Entry Allocation Size	--	Number of URB entries allocated to a Fixed Function unit.
URB Fence	Fence	Virtual, movable boundaries between the URB regions owned by each FF unit.
URB Handle	--	A unique identifier for an URB entry that is passed down a pipeline.
URB Reference	--	
Variable Length Decode	VLD	The first stage of the video decoding pipe that consists mainly of bit-wide operations.
Vertex Buffer	VB	Buffer in memory containing vertex attributes.
Vertex Cache	VC	Cache of Vertex URB Entry (VUE) handles tagged with vertex indices. See the VS chapter for details on this cache.
Vertex Fetcher	VF	The first FF unit in the 3D Pipeline responsible for fetching vertex data from memory. Sometimes referred to as the Vertex Formatter.
Vertex Header	--	Vertex data required for every vertex appearing at the beginning of a Vertex URB Entry.
Vertex ID	--	Unique ID for each vertex that can optionally be included in vertex attribute data sent down the pipeline and used by kernel/shader threads.
Vertex Shader	VS	An API-supplied program that calculates vertex attributes. Also refers to the FF unit that dispatches threads to “shade” (calculate attributes for) vertices.
Vertex URB Entry	VUE	An URB entry that contains data for a specific vertex.
Vertical Stride	VertStride	The distance in element-sized units between 2 vertically-adjacent elements of a region-based GRF access.
Video Front End	VFE	The first fixed function in the generic pipeline; performs fixed-function media operations.

Term	Abbr.	Definition
Viewport	VP	
Windower IZ	WIZ	Term for Windower/Masker that encapsulates its early (“intermediate”) depth test function.
Windower/Masker	WM	Fixed function triangle/line rasterizer.
Word	W	A numerical data type of 16 bits, W represents a signed word integer.

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 2a: Command Reference: Instructions (Command Opcodes)

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

3DPRIMITIVE 1
3DSTATE_AA_LINE_PARAMETERS 6
3DSTATE_BINDING_TABLE_EDIT_DS 8
3DSTATE_BINDING_TABLE_EDIT_GS 10
3DSTATE_BINDING_TABLE_EDIT_HS 12
3DSTATE_BINDING_TABLE_EDIT_PS 14
3DSTATE_BINDING_TABLE_EDIT_VS 16
3DSTATE_BINDING_TABLE_POINTERS_DS 18
3DSTATE_BINDING_TABLE_POINTERS_GS 20
3DSTATE_BINDING_TABLE_POINTERS_HS 22
3DSTATE_BINDING_TABLE_POINTERS_PS 24
3DSTATE_BINDING_TABLE_POINTERS_VS 26
3DSTATE_BINDING_TABLE_POOL_ALLOC 28
3DSTATE_BLEND_STATE_POINTERS 30
3DSTATE_CC_STATE_POINTERS 31
3DSTATE_CHROMA_KEY 32
3DSTATE_CLEAR_PARAMS 34
3DSTATE_CLIP 36
3DSTATE_CONSTANT_DS 42
3DSTATE_CONSTANT_GS 44
3DSTATE_CONSTANT_HS 46
3DSTATE_CONSTANT_PS 48
3DSTATE_CONSTANT_VS 50
3DSTATE_DEPTH_BUFFER 52
3DSTATE_DRAWING_RECTANGLE 61
3DSTATE_DS 64
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC 72
3DSTATE_DX9_CONSTANTB_PS 74
3DSTATE_DX9_CONSTANTB_VS 76
3DSTATE_DX9_CONSTANTF_PS 78
3DSTATE_DX9_CONSTANTF_VS 80

3DSTATE_DX9_CONSTANTI_PS 82
3DSTATE_DX9_CONSTANTI_VS 84
3DSTATE_DX9_GENERATE_ACTIVE_PS 86
3DSTATE_DX9_GENERATE_ACTIVE_VS 89
3DSTATE_DX9_LOCAL_VALID_PS 92
3DSTATE_DX9_LOCAL_VALID_VS 93
3DSTATE_GATHER_CONSTANT_DS 94
3DSTATE_GATHER_CONSTANT_GS 97
3DSTATE_GATHER_CONSTANT_HS 100
3DSTATE_GATHER_CONSTANT_PS 103
3DSTATE_GATHER_CONSTANT_VS 107
3DSTATE_GATHER_POOL_ALLOC 111
3DSTATE_GS 113
3DSTATE_HIER_DEPTH_BUFFER 124
3DSTATE_HS 127
3DSTATE_INDEX_BUFFER 134
3DSTATE_LINE_STIPPLE 136
3DSTATE_MONOFILTER_SIZE 138
3DSTATE_MULTISAMPLE 140
3DSTATE_POLY_STIPPLE_OFFSET 142
3DSTATE_POLY_STIPPLE_PATTERN 144
3DSTATE_PS_BLEND 145
3DSTATE_PS 147
3DSTATE_PS_EXTRA 154
3DSTATE_PUSH_CONSTANT_ALLOC_DS 158
3DSTATE_PUSH_CONSTANT_ALLOC_GS 160
3DSTATE_PUSH_CONSTANT_ALLOC_HS 162
3DSTATE_PUSH_CONSTANT_ALLOC_PS 164
3DSTATE_PUSH_CONSTANT_ALLOC_VS 166
3DSTATE_RASTER 168
3DSTATE_RS_CONSTANT_POINTER 173
3DSTATE_SAMPLE_MASK 175
3DSTATE_SAMPLE_PATTERN 177

3DSTATE_SAMPLER_PALETTE_LOAD0 187
3DSTATE_SAMPLER_PALETTE_LOAD1 188
3DSTATE_SAMPLER_STATE_POINTERS_DS 190
3DSTATE_SAMPLER_STATE_POINTERS_GS 191
3DSTATE_SAMPLER_STATE_POINTERS_HS 192
3DSTATE_SAMPLER_STATE_POINTERS_PS 193
3DSTATE_SAMPLER_STATE_POINTERS_VS 194
3DSTATE_SBE 195
3DSTATE_SBE_SWIZ 203
3DSTATE_SCISSOR_STATE_POINTERS 205
3DSTATE_SF 206
3DSTATE_SO_BUFFER 211
3DSTATE_SO_DECL_LIST 214
3DSTATE_STENCIL_BUFFER 217
3DSTATE_STREAMOUT 220
3DSTATE_TE 225
3DSTATE_URB_CLEAR 229
3DSTATE_URB_DS 231
3DSTATE_URB_GS 233
3DSTATE_URB_HS 235
3DSTATE_URB_VS 237
3DSTATE_VERTEX_BUFFERS 239
3DSTATE_VERTEX_ELEMENTS 241
3DSTATE_VF_COMPONENT_PACKING 243
3DSTATE_VF 247
3DSTATE_VF_INSTANCING 249
3DSTATE_VF_SGVS 251
3DSTATE_VF_STATISTICS 254
3DSTATE_VF_TOPOLOGY 255
3DSTATE_VIEWPORT_STATE_POINTERS_CC 256
3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP 257
3DSTATE_VS 258
3DSTATE_WM_CHROMAKEY 266

3DSTATE_WM_DEPTH_STENCIL 267
3DSTATE_WM 271
3DSTATE_WM_HZ_OP 277
A64 Byte Scaled Read MSD 283
A64 Byte Scaled Write MSD 285
A64 Byte Scattered Read MSD 287
A64 Byte Scattered Write MSD 288
A64 Dword Scaled Read MSD 289
A64 Dword Scaled Write MSD 291
A64 Dword Scattered Read MSD 293
A64 Dword Scattered Write MSD 294
A64 Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD 295
A64 Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD 297
A64 Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD 299
A64 Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD 301
A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD 303
A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 304
A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD 305
A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 306
A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD 307
A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 308
A64 Dword Untyped Atomic Float Binary with Return Data Operation MSD 309
A64 Dword Untyped Atomic Float Binary Write Only Operation MSD 311
A64 Dword Untyped Atomic Float Trinary with Return Data Operation MSD 313
A64 Dword Untyped Atomic Float Trinary Write Only Operation MSD 315
A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD 317
A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD 318
A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD 319
A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD 320
A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD 321
A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD 322
A64 Hword Block Read MSD 323
A64 Hword Block Write MSD 324

A64 Oword Block Read MSD 325
A64 Oword Block Write MSD 326
A64 Oword Dual Block Read MSD 327
A64 Oword Dual Block Write MSD 328
A64 Oword Unaligned Block Read MSD 329
A64 Qword Scaled Read MSD 330
A64 Qword Scaled Write MSD 332
A64 Qword Scattered Read MSD 334
A64 Qword Scattered Write MSD 335
A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD336
A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 337
A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD338
A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 339
A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD340
A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 341
A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD 342
A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD 343
A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD 344
A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD 345
A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD 346
A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD 347
A64 Scaled Untyped Surface Read MSD 348
A64 Scaled Untyped Surface Write MSD 349
A64 Untyped Surface Read MSD 350
A64 Untyped Surface Write MSD 351
Addition 352
Addition with Carry 354
Arithmetic Shift Right 355
Average 357
Bit Field Extract 358
Bit Field Insert 1 362
Bit Field Insert 2 364
Bit Field Reverse 368

Branch Converging 369
Branch Diverging 371
Break 373
Byte Scaled Read MSD 374
Byte Scaled Write MSD 375
Byte Scattered Read MSD 376
Byte Scattered Write MSD 378
Call 380
Call Absolute 382
Compare 384
Compare NaN 386
Conditional Select 388
Conditional Send Message 391
Conditional Split Send Message 393
Constant Cache Dword Scattered Read MSD 396
Constant Cache Oword Block Read MSD 398
Constant Cache Oword Dual Block Read MSD 399
Constant Cache Oword Unaligned Block Read MSD 400
Continue 401
Count Bits Set 403
Dot Product 2 404
Dot Product 3 406
Dot Product 4 408
Dot Product Homogeneous 410
Dword Atomic Counter Binary with Return Data Operation MSD 412
Dword Atomic Counter Binary Write Only Operation MSD 413
Dword Atomic Counter Unary with Return Data Operation MSD 414
Dword Atomic Counter Unary Write Only Operation MSD 415
Dword Scattered Read MSD 416
Dword Scattered Write MSD 418
Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD 420
Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD 421
Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD 422

Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD 423
Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD..... 424
Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD..... 425
Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD 426
Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD 427
Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD 428
Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD 429
Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD 430
Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD 432
Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD..... 434
Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD 436
Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD 438
Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 439
Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD... 440
Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 441
Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD 442
Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 443
Dword Typed Atomic Integer Binary with Return Data Operation MSD..... 444
Dword Typed Atomic Integer Binary Write Only Operation MSD..... 445
Dword Typed Atomic Integer Trinary with Return Data Operation MSD 446
Dword Typed Atomic Integer Trinary Write Only Operation MSD..... 447
Dword Typed Atomic Integer Unary with Return Data Operation MSD..... 448
Dword Typed Atomic Integer Unary Write Only Operation MSD..... 449
Dword Untyped Atomic Float Binary with Return Data Operation MSD 450
Dword Untyped Atomic Float Binary Write Only Operation MSD..... 452
Dword Untyped Atomic Float Trinary with Return Data Operation MSD 454
Dword Untyped Atomic Float Trinary Write Only Operation MSD 456
Dword Untyped Atomic Integer Binary with Return Data Operation MSD 458
Dword Untyped Atomic Integer Binary Write Only Operation MSD..... 459
Dword Untyped Atomic Integer Trinary with Return Data Operation MSD 460
Dword Untyped Atomic Integer Trinary Write Only Operation MSD 461
Dword Untyped Atomic Integer Unary with Return Data Operation MSD 462
Dword Untyped Atomic Integer Unary Write Only Operation MSD 463

Else 464
End If 466
Extended Math Function 468
Find First Bit from LSB Side 472
Find First Bit from MSB Side 474
Fraction 476
Goto 477
GPGPU_CSR_BASE_ADDRESS 479
GPGPU_WALKER 481
Half Precision HI8DS Render Target Write MSD 485
Half Precision LO8DS Render Target Write MSD 488
Half Precision REP16 Render Target Write MSD 491
Half Precision SIMD8 Render Target Write MSD 494
Half Precision SIMD16 Render Target Write MSD 497
Halt 500
HCP_BSD_OBJECT 502
HCP_FQM_STATE 504
HCP_IND_OBJ_BASE_ADDR_STATE 507
HCP_PAK_INSERT_OBJECT 510
HCP_PAK_OBJECT 514
HCP_PIC_STATE 516
HCP_PIPE_BUF_ADDR_STATE 532
HCP_PIPE_MODE_SELECT 538
HCP_QM_STATE 542
HCP_REF_IDX_STATE 545
HCP_SLICE_STATE 548
HCP_SURFACE_STATE 559
HCP_TILE_STATE 562
HCP_VP9_PAK_OBJECT 566
HCP_VP9_PIC_STATE 568
HCP_VP9_SEGMENT_STATE 583
HCP_WEIGHTOFFSET_STATE 587
HEVC_VP9_RDOQ_STATE 590

HI8DS Render Target Write MSD 594
HUC_CFG_STATE 597
HUC_DMEM_STATE 599
HUC_IMEM_STATE 601
HUC_IND_OBJ_BASE_ADDR_STATE 603
HUC_PIPE_MODE_SELECT 605
HUC_START 607
HUC_STREAM_OBJECT 608
HUC_VIRTUAL_ADDR_STATE 611
If 612
Illegal 614
Integer Subtraction with Borrow 615
Join 616
Jump Indexed 618
Leading Zero Detection 620
Line 621
Linear Interpolation 623
LO8DS Render Target Write MSD 626
Logic And 629
Logic Not 631
Logic Or 633
Logic Xor 635
MEDIA_CURBE_LOAD 637
MEDIA_INTERFACE_DESCRIPTOR_LOAD 639
MEDIA_OBJECT 641
MEDIA_OBJECT_GRPID 645
MEDIA_OBJECT_PRT 649
MEDIA_OBJECT_WALKER 651
MEDIA_STATE_FLUSH 658
MEDIA_VFE_STATE 660
Media Block Read MSD 667
Media Block Write MSD 668
Media Transpose Read MSD 669

Memory Fence MSD 670
MFC_AVC_PAK_OBJECT 672
MFC_JPEG_ HUFF_TABLE_STATE 674
MFC_JPEG_SCAN_OBJECT 676
MFC_MPEG2_PAK_OBJECT 679
MFC_MPEG2_SLICEGROUP_STATE 681
MFD_AVC_BSD_OBJECT 689
MFD_AVC_DPB_STATE 692
MFD_AVC_PICID_STATE 695
MFD_AVC_SLICEADDR 697
MFD_IT_OBJECT 700
MFD_JPEG_BSD_OBJECT 703
MFD_MPEG2_BSD_OBJECT 706
MFD_VC1_BSD_OBJECT 708
MFD_VC1_LONG_PIC_STATE 711
MFD_VC1_SHORT_PIC_STATE 724
MFD_VP8_BSD_OBJECT 733
MFX_AVC_DIRECTMODE_STATE 740
MFX_AVC_IMG_STATE 745
MFX_AVC_REF_IDX_STATE 765
MFX_AVC_SLICE_STATE 768
MFX_AVC_WEIGHTOFFSET_STATE 782
MFX_BSP_BUF_BASE_ADDR_STATE 785
MFX_DBK_OBJECT 791
MFX_FQM_STATE 798
MFX_IND_OBJ_BASE_ADDR_STATE 800
MFX_JPEG_HUFF_TABLE_STATE 812
MFX_JPEG_PIC_STATE 814
MFX_MPEG2_PIC_STATE 821
MFX_PAK_INSERT_OBJECT 835
MFX_PIPE_BUF_ADDR_STATE 839
MFX_PIPE_MODE_SELECT 864
MFX_QM_STATE 870

MFX_STATE_POINTER 872
MFX_STITCH_OBJECT 874
MFX_SURFACE_STATE 876
MFX_VC1_DIRECTMODE_STATE 884
MFX_VC1_PRED_PIPE_STATE 888
MFX_VP8_BSP_BUF_BASE_ADDR_STATE 894
MFX_VP8_Encoder_CFG 908
MFX_VP8_PAK_OBJECT 921
MFX_VP8_PIC_STATE 923
MFX_WAIT 952
MI_ARB_CHECK 953
MI_ARB_CHECK 954
MI_ARB_CHECK 955
MI_ARB_CHECK 956
MI_ARB_ON_OFF 957
MI_ATOMIC 959
MI_BATCH_BUFFER_END 964
MI_BATCH_BUFFER_END 965
MI_BATCH_BUFFER_END 966
MI_BATCH_BUFFER_END 967
MI_BATCH_BUFFER_START 968
MI_CLFLUSH 972
MI_CONDITIONAL_BATCH_BUFFER_END 974
MI_COPY_MEM_MEM 977
MI_COPY_MEM_MEM 979
MI_COPY_MEM_MEM 981
MI_COPY_MEM_MEM 983
MI_DISPLAY_FLIP 985
MI_FLUSH_DW 990
MI_FLUSH_DW 993
MI_FLUSH_DW 996
MI_FORCE_WAKEUP 999
MI_LOAD_REGISTER_IMM 1001

MI_LOAD_REGISTER_MEM 1003
MI_LOAD_REGISTER_REG 1005
MI_LOAD_SCAN_LINES_EXCL 1007
MI_LOAD_SCAN_LINES_EXCL 1009
MI_LOAD_SCAN_LINES_INCL 1011
MI_LOAD_SCAN_LINES_INCL 1013
MI_LOAD_URB_MEM 1015
MI_MATH 1016
MI_MATH 1017
MI_MATH 1018
MI_MATH 1019
MI_NOOP 1020
MI_NOOP 1021
MI_NOOP 1022
MI_NOOP 1023
MI_PREDICATE 1024
MI_REPORT_HEAD 1026
MI_REPORT_HEAD 1027
MI_REPORT_HEAD 1028
MI_REPORT_HEAD 1029
MI_REPORT_PERF_COUNT 1030
MI_RS_CONTEXT 1032
MI_RS_CONTROL 1033
MI_RS_STORE_DATA_IMM 1035
MI_SEMAPHORE_SIGNAL 1037
MI_SEMAPHORE_WAIT 1040
MI_SET_CONTEXT 1043
MI_SET_PREDICATE 1046
MI_STORE_DATA_IMM 1048
MI_STORE_DATA_INDEX 1051
MI_STORE_DATA_INDEX 1053
MI_STORE_DATA_INDEX 1055
MI_STORE_DATA_INDEX 1057

MI_STORE_REGISTER_MEM 1059
MI_STORE_URB_MEM 1062
MI_SUSPEND_FLUSH 1064
MI_SUSPEND_FLUSH 1065
MI_SUSPEND_FLUSH 1066
MI_SUSPEND_FLUSH 1067
MI_TOPOLOGY_FILTER 1068
MI_UPDATE_GTT 1069
MI_URB_ATOMIC_ALLOC 1071
MI_USER_INTERRUPT 1072
MI_USER_INTERRUPT 1073
MI_USER_INTERRUPT 1074
MI_USER_INTERRUPT 1075
MI_WAIT_FOR_EVENT 1076
Move 1081
Move Indexed 1083
Multiply 1085
Multiply Accumulate 1087
Multiply Accumulate High 1089
Multiply Add 1091
Multiply Add for Macro 1095
No Operation 1098
Oword Aligned Block Read MSD 1099
Oword Block Read MSD 1100
Oword Block Write MSD 1101
Oword Dual Block Read MSD 1102
Oword Dual Block Write MSD 1103
PIPE_CONTROL 1104
PIPELINE_SELECT 1112
Plane 1115
Read Surface Info MSD 1117
REP16 Render Target Write MSD 1118
Return 1121

Round Down 1123
Round to Nearest or Even 1124
Round to Zero 1126
Round Up 1127
Sampler Cache Media Block Read MSD 1128
Sampler Cache Oword Unaligned Block Read MSD 1129
Scaled Untyped Surface Read MSD 1130
Scaled Untyped Surface Write MSD 1131
Scattered Move 1132
Scratch Block Read MSD 1134
Scratch Block Write MSD 1136
Select 1137
Send Message 1139
SFC_AVS_CHROMA_Coeff_Table 1142
SFC_AVS_LUMA_Coeff_Table 1145
SFC_AVS_STATE 1150
SFC_FRAME_START 1152
SFC_IEF_STATE 1153
SFC_LOCK 1165
SFC_STATE 1167
Shift Left 1192
Shift Right 1194
SIMD8 Render Target Read MSD 1196
SIMD8 Render Target Write MSD 1199
SIMD16 Render Target Read MSD 1202
SIMD16 Render Target Write MSD 1205
Split Send Message 1208
STATE_BASE_ADDRESS 1213
STATE_PREFETCH 1221
STATE_SIP 1223
Sum of Absolute Difference 2 1224
Sum of Absolute Difference Accumulate 2 1226
Typed Surface Read MSD 1228

Typed Surface Write MSD 1229
Untyped Surface Read MSD 1230
Untyped Surface Write MSD 1231
URB Hword Dual Block Read MSD 1232
URB Hword Dual Block Write MSD 1233
URB Oword Block Write MSD 1234
URB Oword Dual Block Read MSD 1235
URB Oword Dual Block Write MSD 1236
VD_PIPELINE_FLUSH 1237
VEB_DI_IECP 1239
VEBOX_STATE 1248
VEBOX_SURFACE_STATE 1256
VEBOX_TILING_CONVERT 1264
Wait Notification 1266
While 1268
XY_COLOR_BLT 1270
XY_FAST_COPY_BLT 1272
XY_FULL_BLT 1275
XY_FULL_IMMEDIATE_PATTERN_BLT 1278
XY_FULL_MONO_PATTERN_BLT 1281
XY_FULL_MONO_PATTERN_MONO_SRC_BLT 1285
XY_FULL_MONO_SRC_BLT 1288
XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT 1291
XY_MONO_PAT_BLT 1294
XY_MONO_PAT_FIXED_BLT 1297
XY_MONO_SRC_COPY_BLT 1300
XY_MONO_SRC_COPY_IMMEDIATE_BLT 1303
XY_PAT_BLT 1306
XY_PAT_BLT_IMMEDIATE 1309
XY_PAT_CHROMA_BLT 1311
XY_PAT_CHROMA_BLT_IMMEDIATE 1313
XY_PIXEL_BLT 1316
XY_SCANLINES_BLT 1317

XY_SETUP_BLT 1318
XY_SETUP_CLIP_BLT 1321
XY_SETUP_MONO_PATTERN_SL_BLT 1322
XY_SRC_COPY_BLT 1325
XY_SRC_COPY_CHROMA_BLT 1328
XY_TEXT_BLT 1331
XY_TEXT_IMMEDIATE_BLT 1333

3DPRIMITIVE

3DPRIMITIVE

Source: RenderCS

Length Bias: 2

The 3DPRIMITIVE command is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in rendering pixel data into the render targets, but this is not required.
The parameters passed in this command are forwarded to the Vertex Fetch function. The Vertex Fetch function will use this information to generate vertex data structures and store them in the URB. These vertices are then passed down the 3D pipeline.

Programming Notes

If the threads spawned by this command are required to observe memory writes performed by threads spawned from a previous command, software must precede this command with a command that performs a (preferably pipelined) memory flush (e.g., 3D_PIPECONTROL).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 3h 3DPRIMITIVE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h 3DPRIMITIVE Format: OpCode
	15	**Reserved**
	14	**Reserved** Format: MBZ
	13	**Reserved** Format: MBZ
	12	**Reserved**
	11	**Reserved**
	10	**Indirect Parameter Enable** Format: Enable If set, the values in DW 2-5 are ignored and replaced by the current values of the corresponding

3DPRIMITIVE

		3DPRIM_xxx MMIO registers: • 3DPRIM_VERTEX_COUNT (instead of DW2: VertexCountPerInstance) • 3DPRIM_START_VERTEX (instead of DW3: StartVertexLocation) • 3DPRIM_INSTANCE_COUNT (instead of DW4: InstanceCount) • 3DPRIM_START_INSTANCE (instead of DW5: StartInstanceLocation) • 3DPRIM_BASE_VERTEX (instead of DW6: BaseVertexLocation) Indirect Parameter Enable and End Offset Enable shall not be ENABLED at the same time, or behavior is UNDEFINED.
	9	**UAV Coherency Required** Format: U1 SW will be required to set this bit if there is the possibility of sharing a UAV from a previous 3DPRIMITVE command. If set, this command may cause a flush due to UAV coherency requirements. If none of the shaders have UAV access enabled, then this bit is ignored.
	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.
	7:0	**DWord Length** Default Value: 5h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:10	**Reserved** Format: MBZ
	9	**End Offset Enable** Format: Enable If set, the Vertex Count Per Instance field is IGNORED, and the 3DPRIM_END_OFFSET register is used to indirectly specify the vertex count by defining the amount of valid data in VB0. The following restrictions apply: • VB0 must be enabled for use • VertexAccessType = SEQUENTIAL • Start Vertex Location = 0 • Start Instance Location = 0 • Base Vertex Location = 0 Vertices are output until EndOffset is reached or exceeded in VB0. If EndOffset is reached or

3DPRIMITIVE

exceeded within the data associated with a vertex, that vertex is considered incomplete and will not be output. Partial objects will be discarded (as is normally done).
If clear, End Offset is ignored.
Indirect Parameter Enable and End Offset Enable must not be ENABLED at the same time, or behavior is UNDEFINED.

8 Vertex Access Type

This field specifies how data held in vertex buffers marked as VERTEXDATA is accessed by Vertex Fetch.

Value	Name	Description
0h	SEQUENTIAL	VERTEXDATA buffers are accessed sequentiallyRequiref if End Offset Enable is ENABLED.
1h	RANDOM	VERTEXDATA buffers are accessed randomly via an index obtained from the Index Buffer.

7:6 Reserved

Format:	MBZ

5:0 Primitive Topology Type

Format:	3D_Prim_Topo_Type See table below for encoding, see 3D Overview for diagrams and general comments

Description
This field specifies the topology type of 3D primitive generated by this command. Note that a single primitive topology (list/strip/fan/etc.) can contain a number of basic objects (lines, triangles, etc.).
This field is ignored. The topology type is specified via the 3DSTATE_VF_TOPOLOGY command.

2 31:0 Vertex Count Per Instance

Format:	U32 Count of vertices

This field specifies how many vertices are to be generated for each instance of the primitive topology. If End Offset Enable is clear:
Format = U32 count of vertices
Range = [0, 2^32-1] (upper limit probably constrained by VB size)
Ignored if End Offset Enable or Indirect Parameter Enable is ENABLED.

Programming Notes

- This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline.
- A 0 value is this field effectively makes the command a 'no-operation'.

3DPRIMITIVE

3	31:0	**Start Vertex Location** Format: U32 structure index This field specifies the "starting vertex" for each instance. This allows skipping over part of the vertices in a buffer if, for example, a previous 3DPRIMITIVE command had already drawn the primitives associated with the earlier entries. For SEQUENTIAL access, this field specifies, for each instance, a starting structure index into the vertex buffers For RANDOM access, this field specifies, for each instance, a starting index into the Index Buffer. **Programming Notes** • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED
4	31:0	**Instance Count** Format: U32 Count of instances This field specifies the number of instances by which the primitive topology is to be regenerated. A value of 0 indicates "no instances" (no-op operation). A value of 1 effectively specifies "non-instanced" operation, though vertex buffers will still be used to provide instance data, if so programmed. Ignored if Indirect Parameter Enable is ENABLED. **Value**: [0, FFFFFFFFh] — **Name**:
5	31:0	**Start Instance Location** Format: U32 structure index **Description** This field specifies the "starting instance" for the command as an initial structure index into Vertex Buffers for vertex elements with InstancingEnable set. Subsequent instances will access sequential instance data structures, as controlled by the Instance Data Step Rate. **Programming Notes** • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED.

<table>
<tr><th colspan="3">3DPRIMITIVE</th></tr>
<tr><td>6</td><td>31:0</td><td>Base Vertex Location
<table><tr><td>Format:</td><td>S31 index structure bias</td></tr></table>This field specifies a signed bias to be added to values read from the index buffer. This allows the same index buffer values to access different vertex data for different commands.This field applies only to RANDOM access mode. This field is ignored for SEQUENTIAL access mode, where there Start Vertex Location can be used to specify different regions in the vertex buffers.
Programming Notes
• Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).
• Must be set to 0 if End Offset Enable is ENABLED.
• Ignored if Indirect Parameter Enable is ENABLED.</td></tr>
</table>

3DSTATE_AA_LINE_PARAMETERS

3DSTATE_AA_LINE_PARAMETERS

Source: RenderCS

Length Bias: 2

The 3DSTATE_AA_LINE_PARAMS command is used to specify the slope and bias terms used in the improved alpha coverage computation (specifically for DX WHQL compliance). Note that in these devices the coverage values passed to PS threads are full U0.8 values, versus in the previous iterations where U0.4 values were passed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_AA_LINE_PARAMETERS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:24	**AA Point Coverage Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.
	23:16	**AA Coverage Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.
	15:8	**AA Point Coverage Slope** Format: U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 0 and 3.If this field is zero, the Windower will revert to legacy aa line coverarge computation (though still output expanded U0.8 coverage values).

3DSTATE_AA_LINE_PARAMETERS		
	7:0	**AA Coverage Slope** Format: U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 0 and 3.If this field is zero, the Windower will revert to legacy aa line coverarge computation (though still output expanded U0.8 coverage values).
2	31:24	**AA Point Coverage EndCap Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 1 and 2.
	23:16	**AA Coverage EndCap Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 1 and 2.
	15:8	**AA Point Coverage EndCap Slope** Format: U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 1 and 2.
	7:0	**AA Coverage EndCap Slope** Format: U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 1 and 2.

3DSTATE_BINDING_TABLE_EDIT_DS

3DSTATE_BINDING_TABLE_EDIT_DS

Source: RenderCS

Length Bias: 2

This command edits the binding table for DS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 46h 3DSTATE_BINDING_TABLE_EDIT_DS Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Value: 0h — Name: DWORD_COUNT_n **[Default]** Value: 0h - 100h — Name: Range
1	31:16	**Binding Table Block Clear** Format: U16 Each bit in this field corresponds to a 16 entry block of the binding table. Bit 0 of this field corresponds to entries 0-15, bit 1 to 16-31, and so on. When a bit is set it clears the corresponding bind table entries to 0. (effectively disabling them). The clear is applied before the individual binding table entries contained in this message are applied. When this bit is clear then the corresponding 16 entry block is not cleared.
	15:2	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_EDIT_DS

	1:0	**Binding Table Edit Target** Specifies which core should respond to this **3DSTATE_BINDING_TABLE_EDIT_DS** command:

Value	Name	Description
11b	All Cores	All cores should respond to this command
10b	Core 1	Only Core1 should respond to this command
01b	Core 0	Only Core0 should respond to this command
00b	Reserved	Reserved

2..n	31:0	**Entry [n]**

Format:	BINDING_TABLE_EDIT_ENTRY

3DSTATE_BINDING_TABLE_EDIT_GS

3DSTATE_BINDING_TABLE_EDIT_GS

Source: RenderCS

Length Bias: 2

This command edits the binding table for GS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 44h 3DSTATE_BINDING_TABLE_EDIT_GS Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Value: 0h — Name: DWORD_COUNT_n **[Default]** Value: 0h - 100h — Name: Range
1	31:16	**Binding Table Block Clear** Format: U16 Each bit in this field corresponds to a 16 entry block of the binding table. Bit 0 of this field corresponds to entries 0-15, bit 1 to 16-31, and so on. When a bit is set it clears the corresponding bind table entries to 0. (effectively disabling them). The clear is applied before the individual binding table entries contained in this message are applied. When this bit is clear then the corresponding 16 entry block is not cleared.
	15:2	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_EDIT_GS

	1:0	**Binding Table Edit Target** Specifies which core should respond to this **3DSTATE_BINDING_TABLE_EDIT_GS** command:

Value	Name	Description
11b	All Cores	All cores should respond to this command
10b	Core 1	Only Core1 should respond to this command
01b	Core 0	Only Core0 should respond to this command
00b	Reserved	Reserved

2..n	31:0	**Entry [n]**

Format:	BINDING_TABLE_EDIT_ENTRY

3DSTATE_BINDING_TABLE_EDIT_HS

3DSTATE_BINDING_TABLE_EDIT_HS

Source: RenderCS

Length Bias: 2

This command edits the binding table for HS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 45h 3DSTATE_BINDING_TABLE_EDIT_HS Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Value / Name 0h: DWORD_COUNT_n **[Default]** 0h - 100h: Range
1	31:16	**Binding Table Block Clear** Format: U16 Each bit in this field corresponds to a 16 entry block of the binding table. Bit 0 of this field corresponds to entries 0-15, bit 1 to 16-31, and so on. When a bit is set it clears the corresponding bind table entries to 0. (effectively disabling them). The clear is applied before the individual binding table entries contained in this message are applied. When this bit is clear then the corresponding 16 entry block is not cleared.
	15:2	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_EDIT_HS

	1:0	**Binding Table Edit Target** Specifies which core should respond to this **3DSTATE_BINDING_TABLE_EDIT_HS** command:

Value	Name	Description
11b	All Cores	All cores should respond to this command
10b	Core 1	Only Core1 should respond to this command
01b	Core 0	Only Core0 should respond to this command
00b	Reserved	Reserved

2..n	31:0	**Entry [n]**

Format:	BINDING_TABLE_EDIT_ENTRY

3DSTATE_BINDING_TABLE_EDIT_PS

3DSTATE_BINDING_TABLE_EDIT_PS

Source: RenderCS

Length Bias: 2

This command edits the binding table for PS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 47h 3DSTATE_BINDING_TABLE_EDIT_PS Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Value / Name 0h: DWORD_COUNT_n **[Default]** 0h - 100h: Range
1	31:16	**Binding Table Block Clear** Format: U16 Each bit in this field corresponds to a 16 entry block of the binding table. Bit 0 of this field corresponds to entries 0-15, bit 1 to 16-31, and so on. When a bit is set it clears the corresponding bind table entries to 0. (effectively disabling them). The clear is applied before the individual binding table entries contained in this message are applied. When this bit is clear then the corresponding 16 entry block is not cleared.
	15:2	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_EDIT_PS

	1:0	**Binding Table Edit Target** Specifies which core should respond to this **3DSTATE_BINDING_TABLE_EDIT_PS** command:

Value	Name	Description
11b	All Cores	All cores should respond to this command
10b	Core 1	Only Core1 should respond to this command
01b	Core 0	Only Core0 should respond to this command
00b	Reserved	Reserved

2..n	31:0	**Entry [n]**

Format:	BINDING_TABLE_EDIT_ENTRY

3DSTATE_BINDING_TABLE_EDIT_VS

3DSTATE_BINDING_TABLE_EDIT_VS

Source: RenderCS

Length Bias: 2

This command edits the binding table for VS.
The 3DSTATE_BINDING_TABLE_EDIT_VS is a variable length command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 43h 3DSTATE_BINDING_TABLE_EDIT_VS Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Value / Name 0h: DWORD_COUNT_n **[Default]** 0h - 100h: Range
1	31:16	**Binding Table Block Clear** Format: U16 Each bit in this field corresponds to a 16 entry block of the binding table. Bit 0 of this field corresponds to entries 0-15, bit 1 to 16-31, and so on. When a bit is set it clears the corresponding bind table entries to 0. (affectively disabling them). The clear is applied before the individual binding table entries contained in this message are applied. When this bit is clear then the corresponding 16 entry block is not cleared.
	15:2	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_EDIT_VS

	1:0	**Binding Table Edit Target** Specifies which core should respond to this **3DSTATE_BINDING_TABLE_EDIT_VS** command:

Value	Name	Description
11b	All Cores	All cores should respond to this command
10b	Core 1	Only Core1 should respond to this command
01b	Core 0	Only Core0 should respond to this command
00b	Reserved	Reserved

2..n	31:0	**Entry [n]**

Format:	BINDING_TABLE_EDIT_ENTRY

3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_BINDING_TABLE_POINTERS_DS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BINDING_TABLE_POINTERS_DS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 28h 3DSTATE_BINDING_TABLE_POINTERS_DS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:16	**Reserved** Format: MBZ
	15:5	**Pointer to DS Binding Table** Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When Binding Table Pool is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When Binding Table Pool is enabled Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.

3DSTATE_BINDING_TABLE_POINTERS_DS		
	4:0	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_BINDING_TABLE_POINTERS_GS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BINDING_TABLE_POINTERS_GS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 29h 3DSTATE_BINDING_TABLE_POINTERS_GS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:16	**Reserved** Format: MBZ
	15:5	**Pointer to GS Binding Table** Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When Binding Table Pool is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When Binding Table Pool is enabled Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.

3DSTATE_BINDING_TABLE_POINTERS_GS		
	4:0	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_HS

3DSTATE_BINDING_TABLE_POINTERS_HS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BINDING_TABLE_POINTERS_HS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 27h 3DSTATE_BINDING_TABLE_POINTERS_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:16	**Reserved** Format: MBZ
	15:5	**Pointer to HS Binding Table** Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When Binding Table Pool is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When Binding Table Pool is enabled Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.

3DSTATE_BINDING_TABLE_POINTERS_HS		
	4:0	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_PS

3DSTATE_BINDING_TABLE_POINTERS_PS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BINDING_TABLE_POINTERS_PS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Ah 3DSTATE_BINDING_TABLE_POINTERS_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:16	**Reserved** Format: MBZ
	15:5	**Pointer to PS Binding Table** Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When Binding Table Pool is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When Binding Table Pool is enabled **Description** Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and

3DSTATE_BINDING_TABLE_POINTERS_PS

		the a398758)lignment is **64B**.
		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled and the setting of HW Binding Table Alignment field: If HW Binding Table Pool is **disabled** and the HW Binding Table Alignmnet is not set to 256B, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table Pool is **enabled** and the HW Binding Table Alignment is not set to 256B, the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**. If HW Binding Table Pool is **disabled** and the HW Binding Table Alignmnet is set to 256B, the offset is relative to the **Surface State Base Address** and the alignment is **256B**. If HW Binding Table Pool is **enabled** and the HW Binding Table Alignmnet is set to 256B, the offset is relative to the **Binding Table Pool Base Address** and the alignment is **256B**.
	4:0	**Reserved**
		Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_VS

3DSTATE_BINDING_TABLE_POINTERS_VS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BINDING_TABLE_POINTERS_VS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 26h 3DSTATE_BINDING_TABLE_POINTERS_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:16	**Reserved** Format: MBZ
	15:5	**Pointer to VS Binding Table** Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When Binding Table Pool is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When Binding Table Pool is enabled Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.

3DSTATE_BINDING_TABLE_POINTERS_VS		
	4:0	**Reserved** Format: MBZ

3DSTATE_BINDING_TABLE_POOL_ALLOC

3DSTATE_BINDING_TABLE_POOL_ALLOC

Source: RenderCS

Length Bias: 2

Description

This command sets up the binding table pool for HW generated binding tables.

Programming Notes

When RS is enabled due to a MI_RS_CONTROL or MI_BATCH_BUFFER_START with RS enable bit set, driver must reprogam the 3DSTATE_BINDING_TABLE_POOL_ALLOC to ensure the resource streamer and render engine are in sync with the programming with the command. Otherwise there could be cases where RS sees that the Binding Table Pool is disabled while the render pipeline sees the binding table is enabled in the case the 3DSTATE_BINDING_TABLE_POOL_ALLOC was enabled while RS was off.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_BINDING_TABLE_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 2h — Name: DWORD_COUNT_n **[Default]**
1..2	63:12	**Binding Table Pool Base Address** Format: GraphicsAddress63-12 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

3DSTATE_BINDING_TABLE_POOL_ALLOC

	11	**Binding Table Pool Enable** Format: U1 When this bit is set it enables HW generation of binding tables. When this bit is cleared it disables HW generation of binding tables.
	10	**Reserved**
	9:7	**Reserved**
	6:0	**Surface Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 2 is not programmable and is always zero.
3	31:12	**Binding Table Pool Buffer Size** Format: U20 This field specifies the size of the buffer in 4K pages. Any access which straddle or go past the end of the buffer will return 0.

Value	Name	Description
[0,1048575]		
0	No Valid Data	There is no valid data in the buffer

Restriction

Restriction : Programming size of zero is illegal in the case that the pool is enabled.

	11:0	**Reserved** Format: MBZ

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS

Source: RenderCS

Length Bias: 2

The 3DSTATE_BLEND_STATE_POINTERS command is used to set up the pointers to the color calculator state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 24h 3DSTATE_BLEND_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:6	**Blend State Pointer** Format: DynamicStateOffset[31:6]BLEND_STATE*8 Specifies the 64-byte aligned offset of the BLEND_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Format: MBZ
	0	**Blend State Pointer Valid** Format: Enable This bit, if set, indicates that the BLEND_STATE pointer has changed and new state needs to be fetched.

3DSTATE_CC_STATE_POINTERS

3DSTATE_CC_STATE_POINTERS

Source: RenderCS

Length Bias: 2

The 3DSTATE_CC_STATE_POINTERS command is used to set up the pointers to the color calculator state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Eh 3DSTATE_CC_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:6	**Color Calc State Pointer** Format: DynamicStateOffset[31:6]COLOR_CALC_STATE Specifies the 64-byte aligned offset of the COLOR_CALC_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Format: MBZ
	0	**Color Calc State Pointer Valid** Format: Enable If set, the hardware will fetch the CC state. This bit is context saved and restored so the CC state is considered undefined once this bit is cleared due to the possiblity of the CC state changing between context switches.

3DSTATE_CHROMA_KEY

3DSTATE_CHROMA_KEY

Source: RenderCS

Length Bias: 2

The 3DSTATE_CHROMA_KEY instruction is used to program texture color/chroma-key key values. A table containing four set of values is supported. The ChromaKey Index sampler state variable is used to select which table entry is associated with the map. Texture chromakey functions are enabled and controlled via use of the ChromaKey Enable texture sampler state variable.Texture Color Key (keying on a paletted texture index) is not supported.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CHROMA_KEY Format: Opcode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**ChromaKey Table Index** Format: U2 index Selects which entry in the ChromaKey table is to be loaded Value: [0,3] — Name:
	29:0	**Reserved** Format: MBZ
2	31:0	**ChromaKey Low Value** This field specifies the "low" (minimum) value of the chroma key range. Texel samples are

<table>
<tr><th colspan="3">3DSTATE_CHROMA_KEY</th></tr>
<tr><td></td><td></td><td>considered "matching the key" if each component of the texel falls within the (inclusive) chroma range.See ChromaKey High Value for further format, programming info.</td></tr>
<tr><td>3</td><td>31:0</td><td>ChromaKey High Value
This field specifies the "high" (maximum) value of the chroma key range. Texel samples are considered "matching the key" if each component of the texel falls within the (inclusive) chroma range.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>ChromaKey values are specified using 8-bit channels. When using surface formats with less than 8 bits per channel, the device will expand channels by replicating the required number of MSBs into the LSBs of each channel. Software must account for this conversion when it programs Chromakey Low/High Values (e.g., by performing the same replication).</td></tr>
<tr><td>For channels that do not exist in the actual surface (e.g., Alpha channel for non-ARGB maps), software must explicitly program full range high/low values (High=FFh, Low=0h for formats using unsigned chroma key values, High=7Fh, Low=FFh for formats using sign magnitude chroma key values) in order to effectively remove the comparison of that field from the ChromaKey function.</td></tr>
<tr><td>For channels in SNORM format in the surface format, the value in the high/low value for that channel is interpreted in sign magnitude format. Negative zero value is not supported (use positive zero instead). For channels with mixed UNORM/SNORM formats (i.e. R5G5_SNORM_B6_UNORM), the ChromaKey is programmed as if all channels are SNORM.</td></tr>
<tr><td>YUV ChromaKey will use an interpolated chrominance value from the map for comparison to the chroma key values for those texels without chrominance due to downsampling. The chrominance value used is the average of values to the left and right of the texel in question.</td></tr>
<tr><td>It is UNDEFINED to program any component of the ChromaKey High Value to be less than the corresponding component of ChromaKey Low Value.</td></tr>
<tr><td>Format = interpreted according to associated texel format "class":</td></tr>
<tr><td>Only the surface formats listed as supported for chroma key in the surface formats table can be used with this feature. Use of any other surface format with chroma key enabled is UNDEFINED.
<table>
<tr><th>Surface Format</th><th>31:24</th><th>23:15</th><th>16:8</th><th>7:0</th></tr>
<tr><td>ARGB and BC (DXT) formats</td><td>A</td><td>R</td><td>G</td><td>B</td></tr>
<tr><td>YCrCb formats</td><td>A</td><td>Cr</td><td>Y</td><td>Cb</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

3DSTATE_CLEAR_PARAMS

3DSTATE_CLEAR_PARAMS

Source: RenderCS

Length Bias: 2

Description

This command defines the depth clear value delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).

HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CLEAR_PARAMS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:0	**Depth Clear Value** Format: IEEE_Float This field defines the clear value that will be applied to the depth buffer if the Depth Buffer Clear field is enabled. It is valid only if Depth Buffer Clear Value Valid is set. **Programming Notes** The clear value must be between the min and max depth values (inclusive) defined in the CC_VIEWPORT. If the depth buffer format is D32_FLOAT, then values must be limited to the range of +0.0f and 1.0f inclusive; values outside this range are reserved.
2	31:1	**Reserved**

3DSTATE_CLEAR_PARAMS		
		Format: MBZ
	0	**Depth Clear Value Valid** Format: Boolean This field enables the **Depth Clear Value**. If clear, the depth clear value is obtained from interpolated depth of an arbitrary pixel of the primitive rendered with **Depth Buffer Clear** set in WM_STATE or 3DSTATE_WM. If set, the depth clear value is obtained from the **Depth Clear Value** field of this command.

3DSTATE_CLIP

3DSTATE_CLIP

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 12h 3DSTATE_CLIP Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 02h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20	**Force User Clip Distance Cull Test Enable Bitmask** Format: Enable This field provides a work around override for the computation of SOL_INT::Render_Enable

Value	Name	Description
0h	Normal	Clip_INT::User Clip Distance Cull Test Enable Bitmask normally
1h	Force	Forces Clip_INT::User Clip Distance Cull Test Enable Bitmask to use the value in 3DSTATE_CLIP:: User Clip Distance Cull Test Enable Bitmask

3DSTATE_CLIP

19 Vertex Sub Pixel Precision Select

Format:	U1

Selects the number of fractional bits maintained in the vertex data

Value	Name	Description
0h	8 Bit	8 sub pixel precision bits maintained
1h	4 Bit	4 sub pixel precision bits maintained

18 Early Cull Enable

Format:	Enable

This field is used to enable/disable the EarlyCull function. When this bit is set triangles are checked if they are backface culled before proceedign through must clip function.

Programming Notes
Setting this bit must not impact functionality, this state only controls the performance of the must clip function. Vertex Sub Pixel Precision Selectprecision must be set to "8 bit" in order avoid precision issues.

17 Force User Clip Distance Clip Test Enable Bitmask

Format:	Enable

This field provides a work around override for the computation of SOL_INT::Render_Enable.

Value	Name	Description
0b	Normal	Clip_INT:: User Clip Distance Clip Test Enable Bitmask normally
1b	Force	Forces Clip_INT:: User Clip Distance Clip Test Enable Bitmask to use the value in 3DSTATE_CLIP::User Clip Distance Clip Test Enable Bitmask

16 Force Clip Mode

Format:	Enable

This field provides a work around override for the computation of SOL_INT::Render_Enable.

Value	Name	Description
0b	Normal	Clip_INT::Clip Mode is computed normally.
1b	Force	Forces Clip_INT::Clip Mode to use the value in 3DSTATE_CLIP::User Clip Mode.

15:11 Reserved

Format:	MBZ

10 Clipper Statistics Enable

Format:	Enable

This bit controls whether Clip-unit-specific statistics register(s) can be incremented.

Value	Name	Description
0h	Disable	CL_INVOCATIONS_COUNT cannot increment
1h	Enable	CL_INVOCATIONS_COUNT can increment

9:8 Reserved

Format:	MBZ

3DSTATE_CLIP

	7:0	**User Clip Distance Cull Test Enable Bitmask** Format: Enable[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip).DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.
2	31	**Clip Enable** Format: Enable Specifies whether the Clip function is enabled or disabled (pass-through).
	30	**API Mode** Controls the definition of the NEAR clipping plane Value / Name / Description 0h / OGL / NEAR VP boundary == 0.0 (NDC)
	29	**Reserved** Format: MBZ
	28	**Viewport XY Clip Test Enable** Format: Enable This field is used to control whether the Viewport X, Y extents [-1,1] are considered in VertexClipTest. If both the Guardband and Viewport XY ClipTest are DISABLED, all vertices are considered "visible" with respect to the XY directions.
	27	**Reserved** Format: MBZ
	26	**Guardband Clip Test Enable** Format: Enable This field is used to control whether the Guardband X, Y extents are considered in VertexClipTest for non-point objects. If the Guardband ClipTest is DISABLED but the Viewport XY ClipTest is ENABLED, ClipDetermination operates as if the Guardband were coincident with the Viewport. If both the Guardband and Viewport XY ClipTest are DISABLED, all vertices are considered "visible" with respect to the XY directions.
	25:24	**Reserved** Format: MBZ
	23:16	**User Clip Distance Clip Test Enable Bitmask** Format: Enable[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made.DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

3DSTATE_CLIP

15:13 Clip Mode

This field specifies a general mode of the CLIP unit, when the CLIP unit is ENABLED.

Value	Name	Description
0h	NORMAL	TrivialAccept objects are passed down the pipeline, MustClip objects Clipped in the Fixed Function Clipper HW, TrivialReject and BAD objects are discarded
1h	Reserved	
2h	Reserved	
3h	REJECT_ALL	All objects are discarded
4h	ACCEPT_ALL	All objects (except BAD objects) are trivially accepted. This effectively disables the clip-test/clip-determination function. Note that the CLIP unit will still filter out adacency information, which may be required since the SF unit does not accept primitives with adjacency.
5h-7h	Reserved	

12:10 Reserved

Format:	MBZ

9 Perspective Divide Disable

Format:	Disable

This field disables the Perspective Divide function performed on homogeneous position read from the URB. This feature can be used by software to submit pre-transformed "screen-space" geometry for rasterization. This likely requires the W component of positions to contain "rhw" (aka 1/w) in order to support perspective-correct interpolation of vertex attributes. Likewise, the X, Y, Z components will likely be required to be X/W, Y/W, Z/W. Note that the device does not support clipping when perspective divide is disabled. Software must specify CLIPMODE_ACCEPT_ALL whenever it disables perspective divide. This implies that software must ensure that object positions are completely contained within the "guardband" screen-space limits imposed by the SF unit (e.g., by clipping in CPU SW before submitting the objects).

8 Non-Perspective Barycentric Enable

Format:	Enable

This field enables computation of non-perspective barycentric parameters in the clipper, which are sent to SF unit in the must clip case. This field must be enabled if any non-perspective interpolation modes are used in pixel shader.

Programming Notes
This field must be set whenever Enable bits 3 or 4 or 5 of 3DSTATE_WM:Barycentric Interpolation Mode is set. This indicates that one of the Non-perspetive barycentric interpolation modes are used.

7:6 Reserved

Format:	MBZ

3DSTATE_CLIP

	5:4	**Triangle Strip/List Provoking Vertex Select** Format: U2 enumerated type This field selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". **Value** / **Name** 0h: 0 1h: 1 2h: 2 3h: Reserved
	3:2	**Line Strip/List Provoking Vertex Select** Format: U2 enumerated type This field selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex". **Value** / **Name** 0h: 0 1h: 1 2h: Reserved 3h: Reserved
	1:0	**Triangle Fan Provoking Vertex Select** Format: U2 enumerated type This field selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex". **Value** / **Name** 0h: 0 1h: 1 2h: 2 3h: Reserved
3	31:28	**Reserved** Format: MBZ

3DSTATE_CLIP

	27:17	**Minimum Point Width** Format: U8.3 pixels This value is used to clamp read-back PointWidth values.
	16:6	**Maximum Point Width** Format: U8.3 pixels This value is used to clamp read-back PointWidth values.
	5	**Force Zero RTA Index Enable** Format: Enable If set, the Clip unit will ignore the read-back RTAIndex and operate as if the value 0 was read-back. If clear, the read-back value is used.
	4	**Reserved** Format: MBZ
	3:0	**Maximum VP Index** Format: U4-1 index value (# of viewports) This field specifies the maximum valid VPIndex value, corresponding to the number of active viewports. If the source of the VPIndex exceeds this maximum value, a VPIndex value of 0 is passed down the pipeline. Note that this clamping does not affect a VPIndex value stored in the URB.

3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT_DS

Source: RenderCS

Length Bias: 2

This command sets pointers to the push constants for the DS unit. The constant data pointed to by this command is loaded into the DS unit's push constant buffer (PCB).

Programming Notes

3DSTATE_GATHER_DS command must be dispatched along with any 3DSTATE_CONSTANT_DS command when Gather Pool is enabled within a batch buffer.

The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed.
On usage model of enabling legacy mode is when Resource Streamer is not enabled.

Workaround

Workaround:
The driver must ensure The following case does not occur without a flush to the 3D engine:
3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a
3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_CONSTANT_DS

	23:16	**3D Command Sub Opcode** Default Value: 1Ah 3DSTATE_CONSTANT_DS Format: OpCode
	15	**Reserved** Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: **MEMORY_OBJECT_CONTROL_STATE** Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** 9h: Excludes DWord (0,1) **[Default]**
1..10	319:0	**Constant Body** Format: **3DSTATE_CONSTANT(Body)** See the 3DSTATE_CONSTANT(Body) format for the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS.

3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT_GS

Source: RenderCS

Length Bias: 2

This command sets pointers to the push constants for the GS unit. The constant data pointed to by this command will be loaded into the GS unit's push constant buffer (PCB).

Programming Notes

A 3DSTATE_GATHER_GS command must be dispatched along with any 3DSTATE_CONSTANT_GS command when the Gather Pool is enabled within a batch buffer.

The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed.
On usage model of enabling legacy mode is when Resource Streamer is not enabled.

Workaround

Workaround:
The driver must ensure The following case does not occur without a flush to the 3D engine:
3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a
3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_CONSTANT_GS

	23:16	**3D Command Sub Opcode** Default Value: 16h 3DSTATE_CONSTANT_GS Format: OpCode
	15	**Reserved** Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]**
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT_HS

Source: RenderCS

Length Bias: 2

This command sets pointers to the push constants for the HS unit. The constant data pointed to by this command is loaded into the HS unit's push constant buffer (PCB).

Programming Notes

A 3DSTATE_GATHER_HS command must be dispatched along with any 3DSTATE_CONSTANT_HS command when Gather Pool is enabled within a batch buffer.

The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed.
On usage model of enabling legacy mode is when Resource Streamer is not enabled.

Workaround

Workaround:
The driver must ensure The following case does not occur without a flush to the 3D engine:
3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a 3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_CONSTANT_HS

	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_CONSTANT_HS Format: OpCode
	15	**Reserved** Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]**
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_PS

3DSTATE_CONSTANT_PS

Source: RenderCS

Length Bias: 2

This command sets pointers to the push constants for the PS unit. The constant data pointed to by this command is loaded into the PS unit's push constant buffer (PCB).

Programming Notes

A 3DSTATE_GATHER_PS command must be dispatched along with any 3DSTATE_CONSTANT_PS command when the Gather Pool is enabled within a batch buffer.

The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed.
On usage model of enabling legacy mode is when Resource Streamer is not enabled.

Workaround

Workaround:
The driver must ensure The following case does not occur without a flush to the 3D engine:
3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a 3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_CONSTANT_PS

	23:16	**3D Command Sub Opcode** Default Value: 17h 3DSTATE_CONSTANT_PS Format: OpCode
	15	**Reserved** Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**Dword Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]**
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_VS

3DSTATE_CONSTANT_VS

Source: RenderCS

Length Bias: 2

This command sets pointers to the push constants for VS unit. The constant data pointed to by this command is loaded into the VS unit's push constant buffer (PCB).

Programming Notes

A 3DSTATE_GATHER_VS command must be dispatched along with any 3DSTATE_CONSTANT_VS command when Gather Pool is enabled within a batch buffer.

The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed.
On usage model of enabling legacy mode is when Resource Streamer is not enabled.

Workaround

Workaround:
The driver must ensure The following case does not occur without a flush to the 3D engine:
3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a 3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_CONSTANT_VS

	23:16	**3D Command Sub Opcode** Default Value: 15h 3DSTATE_CONSTANT_VS Format: OpCode
	15	**Reserved** Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]**
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_DEPTH_BUFFER

3DSTATE_DEPTH_BUFFER

Source: RenderCS

Length Bias: 2

The depth buffer surface state is delivered as a pipelined state packet. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this commands is issued. The PIPE_CONTROL restrictions are removed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 5h 3DSTATE_DEPTH_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Excludes DWord(0,1) Value: 6h — Name: Excludes Dword (0,1) **[Default]**
1	31:29	**Surface Type** Value / Name / Description: 0h / Reserved / Reserved 1h / SURFTYPE_2D / Defines a 2-dimensional map or array of maps 2h / Reserved / Reserved 3h / SURFTYPE_CUBE / Defines a cube map 4h-6h / Reserved / 7h / SURFTYPE_NULL / Defines a null surface

3DSTATE_DEPTH_BUFFER

		Programming Notes The Surface Type of the depth buffer must be the same as the Surface Type of the render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL. 1D surface type not allowed for depth surface and stencil surface. **Workaround** Workaround : If depth/stencil is enabled with 1D render target, depth/stencil surface type needs to be set to 2D surface type and height set to 1. Depth will use (legacy) TileY and stencil will use TileW. For this case only, the Surface Type of the depth buffer can be 2D while the Surface Type of the render target(s) are 1D, representing an exception to a programming note above.
	28	**Depth Write Enable** Format: Enable This field enables depth writes to the depth buffer surface. Both this field and the Depth Buffer Write Enable field in DEPTH_STENCIL_STATE must be enabled in order for depth writes to occur.
	27	**Stencil Write Enable** Format: Enable This field enables stencil writes to the depth buffer or stencil buffer surface, depending on where stencil is located. Both this field and the Stencil Buffer Write Enable field in DEPTH_STENCIL_STATE must be enabled in order for stencil writes to occur.
	26:24	**Reserved** Format: MBZ
	23	**Reserved** Format: MBZ
	22	**Hierarchical Depth Buffer Enable** Format: Enable If enabled, indicates that a hierarchical depth buffer is defined. **Programming Notes** If this field is enabled, the Software Tiled Rendering Mode must be NORMAL. This field must be disabled if Early Depth Test Enable is disabled OR if depth buffer surface type is NULL. This field must be disabled if surface type field is SURFTYPE_1D
	21	**Reserved** Format: MBZ
	20:18	**Surface Format** Specifies the format of the depth buffer. See Stencil Test Enable field in

3DSTATE_DEPTH_BUFFER

DEPTH_STENCIL_STATE field for restrictions on the use of some of these formats.

Value	Name
0h	Reserved
1h	D32_FLOAT
2h	Reserved
3h	D24_UNORM_X8_UINT
4h	Reserved
5h	D16_UNORM
6h-7h	Reserved

17:0 **Surface Pitch**

Format:	U18-1 Pitch in (Bytes-1)

Description

For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to *Memory Data Formats* section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1,262143] -> [(2^{Cu})B,256KB] = [1 tile, 256KB/(2^{Cu}) tiles]

Value	Name
[7Fh,3FFFFh]	

Programming Notes

The pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].
The minimum pitch should be calculated as per the formula given below.

pixel_size	Depth Format
2	Z16
4	Z24
4	Z32

The minimum pitch should be calculated based on Cu, Cv, W_L.
The Cu, Cv are the tile constants and $W_L$is the aligned width adjusted for MSAA.
to get the Cu, Cv, $W_L$values and Calculations.
Then use this for pitch formula :
Minimum_pitch =(ceiling((W_0 pixel_size) / (1 « Cu)) *(1 « Cu))– 1 ; //$W_0$is the aligned width for the largest LOD (i.e LOD 0)*
(1 « Cu) = tile width in bytes
(1 « Cv) = tile height in lines

3DSTATE_DEPTH_BUFFER

2..3 This field specifies the address of the buffer in mapped Graphics Memory. **Programming Notes:** The Depth Buffer can only be mapped to Main Memory (uncached). If the surface is tiled, the base address must conform to the Per-Surface Tiling Alignment Rules. If the buffer is linear, the surface must be 64-byte aligned.	63:0	**Surface Base Address** Format: GraphicsAddress63-0
4	31:18	**Height** Format: U14-1 This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level.

Value	Name	Description	Exists If
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,16383]	Legal Range	Height of surface - 1 (y/v dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,16383]	Legal Range	Height of surface - 1 (y/v dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')

3DSTATE_DEPTH_BUFFER

[0,16383]	Legal Range	y/v dimension	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes

The Height of the depth buffer must be the same as the Height of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped).

17:4 **Width**

Format:	U14-1

This field specifies the width of the surface. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels.

Value	Name	Description	Exists If
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes

The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field).

For cube maps, Width must be set equal to Height.

The Width of the depth buffer must be the same as the Width of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped).

3:0 **LOD**

Format:	U4 for LOD units

Value	Name
[0,14]	

Programming Notes

The LOD of the depth buffer must be the same as the LOD of the render target(s) (defined in SURFACE_STATE)

3DSTATE_DEPTH_BUFFER

5 — 31:21 — Depth

Format:	U11-1

This field specifies the total number of levels for a volume texture or the number of array elements allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.

Value	Name	Description	Exists If
[0,2047]	Legal Range	Number of array elements - 1	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,2047]	Legal Range	Number of array elements - 1	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,2047]	Legal Range	Depth of surface - 1 (r/z dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes

The Depth of the depth buffer must be the same as the Depth of the render target(s) (defined in SURFACE_STATE).

20:10 — Minimum Array Element

Format:	U11

For 1D and 2D Surfaces:
This field indicates the minimum array element that can be accessed as part of this surface. The delivered array index is added to this field before being used to address the surface.
For 3D Surfaces
This field indicates the minimum 'R' coordinate on the LOD currently being rendered to. This field is added to the delivered array index before it is used to address the surface.
For Other Surfaces
This field is ignored

Value	Name	Exists If
[0,2047]	SURFTYPE_1D/2D	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D'|Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,2047]	SURFTYPE_3D	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')

9:7 — Reserved

Format:	MBZ

6:0 — Depth Buffer Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for the depth buffer.

3DSTATE_DEPTH_BUFFER

6	31:30	**Tiled Resource Mode** **For Depth Buffer Surfaces:** This field specifies the tiled resource mode. **For other surfaces:** This field is ignored.

Value	Name	Description
0h	NONE	No tiled resource
1h	TILEYF	4KB tiled resources
2h	TILEYS	64KB tiled resources
3h	Reserved	

Programming Notes
If **Tile Mode** is not set to TILEMODE_YMAJOR, this field must be set to TRMODE_NONE.
HIZ and Stencil surfaces can't be tiled resource, hence will continue to follow legacy tiledY and tiledW respectively.

29:26 **Mip Tail Start LOD**

Format:	U4 in LOD Units

For Sampling Engine, Render Target, and Typed Surfaces: This field indicates which LOD is the first one in the MIP tail if **Tiled Mode** is not TRMODE_NONE. The MIP tail has a different layout than the rest of the surface. Refer to the *Memory Data Formats* section for more details. **For other surfaces:** This field is ignored.

Programming Notes
This field must be zero if the **Surface Format** is MONO8
This field is ignored if **Tiled Mode** is TRMODE_NONE unless **Surface Type** is SURFTYPE_1D.
If **Tiled Mode** is not TRMODE_NONE, this field must be set to ensure that mips within the mip tail do not overlap given the storage algorithms given in the Memory Data Formats section. The following table indicates the *maximum* size of the mip that is set to be the Mip Tail Start LOD for various cases:

Surface Type	Tiling Mode	#MS	Bits Per Element				
			8	16	32	64	128
1D	64KB	1	16384	8192	4096	2048	1024
	4KB	1	1024	512	256	128	64
2D/ CUBE	TIleYS	1	128x256	128x128	64x128	64x64	32x64
		2	128x128	64x128	64x64	32x64	32x32
		4	64x128	64x64	32x64	32x32	16x32
		8	64x64	32x64	32x32	16x32	16x16

3DSTATE_DEPTH_BUFFER

		16	32x64	32x32	16x32	16x16	8x16
	TileYF	any	64x32	32x32	32x16	16x16	16x8
3D	TIleYS	1	32x32x32	16x32x32	16x32x16	16x16x16	8x16x16
	TIleYF	1	16x8x16	8x8x16	8x8x8	8x4x8	4x4x8

25:0 Reserved

Format:	MBZ

7 — 31:21 Render Target View Extent

Format:	U11-1

Value	Name	Description	Exists If
[0,2047]	Legal Range	Number of array elements- 1	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,2047]	Legal Range	Number of array elements- 1	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,2047]	Legal Range	To indication extent of [1,2048]	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

20:15 Reserved

14:0 Surface QPitch

Format:	QPitch[16:2]

Description

The interpretation of this field is dependent on **Surface Type** as follows:

- SURFTYPE_1D: distance in *pixels* between array slices
- SURFTYPE_2D/CUBE: distance in *rows* between array slices
- SURFTYPE_3D: distance in *rows* between R-slices

For 1D and 2D Surfaces:
This field must be set to the same value as the Depth field.
For 3D Surfaces:
This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to.
For Other Surfaces
This field is ignored.

Value	Name	Description
[4h, 1FFFCh]		in multiples of 4 (low 2 bits missing)

3DSTATE_DEPTH_BUFFER		
		Programming Notes Software must ensure that this field is set to a value sufficiently large that array slices in the surface do not overlap. Refer to the *Memory Data Formats* section for information on how surfaces are stored. TYS/TYF QPitch is valid only for 2D array surfaces and represents the tile-padded total number of texels(lines) in a single array slice. Height of each LOD: HL = AlignToTileHeight(MSAA_height_factor * (**height** »L) > 0? **height**»L : 1), where AlignToTileHeight(x) is (ceiling((x) / (1 « Cv)) *(1 « Cv)) Height of all LODs is a sum: H = H0+H1+..Hn, N is number of mip levels. If surface has MIP tail, equation stops at Hn where n=MipTailStartLOD. MipTail is single tile. QPitch is multiple of tile height (1 « Cv) and should be equal or greater H computed above.

3DSTATE_DRAWING_RECTANGLE

3DSTATE_DRAWING_RECTANGLE

Source: RenderCS

Length Bias: 2

The 3DSTATE_DRAWING_RECTANGLE command is used to set the 3D drawing rectangle and related state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 00h 3DSTATE_DRAWING_RECTANGLE Format: OpCode
	15:14	**Core Mode Select** Format: U2 Specifies which core this command will be considered valid and update based on the state in this command. Value / Name / Description: 0h / Legacy / Both cores are enabled and will update the state. 1h / Core 0 Enabled / State will be updated in Core 0 only 2h / Core 1 Enabled / State will be updated in Core 1 only 3h / Reserved /
	13:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:16	**Clipped Drawing Rectangle Y Min** Format: U16 in Pixels from Color Buffer origin (upper left corner) Specifies Ymin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with Y coordinates less than Ymin will be clipped out.

3DSTATE_DRAWING_RECTANGLE

Value	Name
[0,16383]	Device ignores bits 31:30

Programming Notes
This value can be larger than Clipped Drawing Rectangle Y Max. If Ymin>Ymax, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.

15:0 **Clipped Drawing Rectangle X Min**

Format:	U16 in Pixels from Color Buffer origin (upper left corner)

Specifies Xmin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with X coordinates less than Xmin will be clipped out.

Value	Name
[0,16383]	Device ignores bits 15:14

Programming Notes
This value can be larger than Clipped Drawing Rectangle X Max. If Xmin>Xmax, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.

2 31:16 **Clipped Drawing Rectangle Y Max**

Format:	U16 in Pixels from Color Buffer origin (upper left corner)

Specifies Ymax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Ymax will be clipped out.

Value	Name
[0,16383]	Device ignores bits 31:30

Programming Notes
This value can be less than Clipped Drawing Rectangle Y Min. If Ymax<Ymin, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.

15:0 **Clipped Drawing Rectangle X Max**

Format:	U16 in Pixels from Color Buffer origin (upper left corner)

Specifies Xmax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Xmax will be clipped out.

Value	Name
[0,16383]	Device ignores bits 15:14

Programming Notes
This value can be less than Clipped Drawing Rectangle X Min. If Xmax<Xmin, the clipped drawing rectangle is null, all polygons are discarded.If Xmin==Xmax, the clipped drawing

3DSTATE_DRAWING_RECTANGLE

		rectangle is 1 pixel wide in the X direction.
3	31:16	**Drawing Rectangle Origin Y** Format: S15 in Pixels from Color Buffer origin (upper left corner). **Description** Range: [-16384,16383] (Bit 31 should be a sign extension) Specifies Y origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.
	15:0	**Drawing Rectangle Origin X** Format: S15 in Pixels from Color Buffer origin (upper left corner). **Description** Range: [-16384,16383] (Bit 15 should be a sign extension) Specifies X origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.

3DSTATE_DS

3DSTATE_DS		
Source:	RenderCS	
Length Bias:	2	
The state used by DS is defined with this inline state packet		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Dh 3DSTATE_DS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:6	**Kernel Start Pointer** Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if DS Function Enable is DISABLED.
	5:0	**Reserved** Format: MBZ
3	31	**Reserved** Format: MBZ

3DSTATE_DS

30 Vector Mask Enable

Format:	U1 Enumerated Type

Upon subsequent DS thread dispatches, this bit is loaded into the EU's Vector Mask Enable (VME, cr0.0[3]) thread state. Refer to EU documentation for the definition and use of VME state.

Value	Name	Description
0h	Dmask	The EU will use the Dispatch Mask (supplied by the DS stage) for instruction execution.
1h	Vmask	The EU will use the Vector Mask (derived from the Dispatch Mask) for instruction execution.

Programming Notes
Under normal conditions SW shall specify DMask, as the DS stage will provide a Dispatch Mask appropriate to SIMD4x2 or SIMD8 thread execution (as a function of dispatch mode). E.g., for SIMD4x2 thread execution, the DS stage will generate a Dispatch Mask that is equal to what the EU would use as the Vector Mask. For SIMD8 execution there is no known usage model for use of Vector Mask (as there is for PS shaders).

29:27 Sampler Count

Format:	U3

Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the associated sampler state entries.
This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description
0h	No Samplers	No samplers used
1h	1-4 Samplers	between 1 and 4 samplers used
2h	5-8 Samplers	between 5 and 8 samplers used
3h	9-12 Samplers	between 9 and 12 samplers used
4h	13-16 Samplers	between 13 and 16 samplers used

26 Reserved

Format:	MBZ

25:18 Binding Table Entry Count

Format:	U8

When HW Generated Binding Table is disabled:
Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.
Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.
This field is ignored if DS Function Enable is DISABLED.

When HW Generated Binding Table bit is enabled:

3DSTATE_DS

This field indicates which cache lines (512bit units - 32 Binding Table Entry section) should be fetched. Each bit in this field corresponds to a cache line. Only the 1st 4 non-zero Binding Table entries of each 32 Binding Table entry section prefetched will have its surface state prefetched.

Value	Name
[0,255]	

Programming Notes
When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be generated at JIT time.

17 **Thread Dispatch Priority**

Format:	U1 Enumerated Type

Specifies the priority of the thread for dispatch:
This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description
0h	Normal	Normal Priority
1h	High	High Priority

16 **Floating Point Mode**

Format:	U1 Enumerated Type

Specifies the initial floating point mode used by the dispatched thread.
This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description
0h	IEEE-754	Use IEEE-754 Rules
1h	Alternate	Use alternate rules

15 **Reserved**

Format:	MBZ

14 **Accesses UAV**

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.

Programming Notes
This field must not be set when DS Function Enable is disabled.

13 **Illegal Opcode Exception Enable**

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.
This field is ignored if DS Function Enable is DISABLED.

12:8 **Reserved**

3DSTATE_DS

		Format: MBZ
	7	**Software Exception Enable** Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if DS Function Enable is DISABLED.
	6:0	**Reserved** Format: MBZ
4..5	63:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[63:10]ScratchSpace Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if DS Function Enable is DISABLED.
	9:4	**Reserved** Format: MBZ
	3:0	**Per-Thread Scratch Space** Format: U4 power of 2 Bytes over 1K Bytes Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit.The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if DS Function Enable is DISABLED. **Value** / **Name**: [0,11] indicating [1K Bytes, 2M Bytes] **Programming Notes**: This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.
6	31:25	**Reserved** Format: MBZ
	24:20	**Dispatch GRF Start Register For URB Data** Format: GRFRegister[4:0] Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.

3DSTATE_DS

This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description
[0,31]		indicating GRF [R0, R31]

Programming Notes
When SIMD8_SINGLE_OR_DUAL_PATCH dispatch mode is selected, SW shall program this field to a value of 6 or greater to accommodate either the DUAL_PATCH or SIMD8_SINGLE_PATCH payloads.

19:18 Reserved

Format:	MBZ

17:11 Patch URB Entry Read Length

Format:	U7

Specifies how much data (in 256-bit units) is to be read from the Patch URB entry and passed in the DS thread payload.
This field is ignored if DS Function Enable is DISABLED.

Value	Name
[0,64]	

10 Reserved

Format:	MBZ

9:4 Patch URB Entry Read Offset

Format:	U6

Specifies the offset (in 256-bit units) at which Patch URB data is to be read from the URB before being included in the thread payload.
This field is ignored if DS Function Enable is DISABLED.

Value	Name
[0,63]	

3:0 Reserved

Format:	MBZ

7 31 Reserved

Format:	MBZ

30 Reserved

Format:	MBZ

29:21 Maximum Number of Threads

Format:	U9-1 Thread Count

Specifies the maximum number of simultaneous DS threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual

3DSTATE_DS

dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance.
This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description
[0,335]		indicating thread count of [1,336]

20:11 Reserved

Format:	MBZ

10 Statistics Enable

Format:	Enable

If ENABLED, this FF unit will engage in statistics gathering. Refer to the Statistics Gathering section. If DISABLED, statistics information associated with this FF stage will be left unchanged.
This field is ignored if DS Function Enable is DISABLED.

9:5 Reserved

Format:	MBZ

4:3 Dispatch Mode

Format:	U2

This field specifies how the DS stage generates DS thread requests, and correspondingly impacts the DS thread payload. The setting of this field must agree with how the DS kernel was compiled.
This field is ignored if DS Function Enable is DISABLED.

Value	Name	Description	Programming Notes
0h	SIMD4X2	DS threads are passed one patch, up to 2 domain point inputs, and up to two output vertex handles. The DS kernel (at KSP) is expected to run in SIMD4x2 execution mode. The DUAL_PATCH KSP is ignored. The **Single Domain Point Dispatch** field can be used to force single domain point dispatches.	
1h	SIMD8_SINGLE_PATCH	DS threads are passed one patch, up to 8 domain point inputs, and up to 8 output vertex handles. All the domain points are associated with the single input patch. The DS kernel (at KSP) is expected to run in SIMD8 execution mode. The DUAL_PATCH KSP is ignored.	

3DSTATE_DS

		2h	SIMD8_SINGLE_OR_DUAL_PATCH	SIMD8_SINGLE_OR_DUAL_PATCH This mode enables use of both the KSP and the DUAL_PATCH KSP. The KSP kernel operates just like in SIMD8_SINGLE_PATCH mode. See DUAL_PATCH Thread Execution for a discussion of how the DUAL_PATCH KSP is used.	At least 2 HS URB handles must be allocated in order to enable this mode. SIMD8_SINGLE_OR_DUAL_PATCH must not be used if the domain shader kernel uses primitive id.
		3h	Reserved		
	2	**Compute W Coordinate Enable** Format: Enable If ENABLED, the DS unit will (for each domain point) compute W = 1 - (U + V) and pass the result as a floating point value in the DS thread payload. If DISABLED, 0.0 will be passed. This field must only be ENABLED for the tessellation of TRI domains, where UVW coordinates are required. This field must be DISABLED for other domains (as they only require UV coordinates) otherwise the computed W coordinate is UNDEFINED. This field is ignored if DS Function Enable is DISABLED.			
	1	**Cache Disable** Format: Disable This bit controls the operation of the DS Cache. This field is ignored if DS Function Enable is DISABLED. If the DS Cache is DISABLED and the DS Function is ENABLED, the DS Cache is not used and all incoming domain points will be passed to DS threads. If the DS Cache is ENABLED and the DS Function is ENABLED, incoming domain points that do not hit in the DS Cache will be passed to DS threads. The DS Cache is invalidated whenever the DS Cache becomes DISABLED , whenever the DS Function Enable toggles, and between patches.			
	0	**Function Enable** Format: Enable If ENABLED, DS threads will be spawned to process incoming domain points which miss in the DS cache. If DISABLED, the DS stage goes into pass-through mode and performs no specific processing. This field is always used. **Programming Notes** The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.			
8	31:28	**Reserved** Format: MBZ			
	27	**Reserved**			

3DSTATE_DS

26:21 Vertex URB Entry Output Read Offset

Format:	U6

Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB by SBE.

Value	Name
[0,63]	

20:16 Vertex URB Entry Output Length

Format:	U5

Specifies the amount of URB data written for each Vertex URB entry, in 256-bit register increments.

Value	Name
[1,16]	

Programming Notes
This length does not include the vertex header.

15:8 User Clip Distance Clip Test Enable Bitmask

Format:	U8

This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made.

DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

7:0 User Clip Distance Cull Test Enable Bitmask

Format:	U8

This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip).

DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

9..10 63:6 DUAL_PATCH Kernel Start Pointer

Format:	InstructionBaseOffset[63:6]Kernel

This field specifies the starting location of the DUAL_PATCH kernel program run by threads spawned by this FF unit.

It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if DS Function Enable is DISABLED.

See DUAL_PATCH Thread Execution for a discussion of how the DUAL_PATCH KSP is used.

5:0 Reserved

Format:	MBZ

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

Source: RenderCS

Length Bias: 2

This command sets up the Gather Pool for Gather Buffers.

Programming Notes

This command must only be programmed when resource streamer is enabled thru batch buffer start and MI_RS_CONTROL has not disabled resource streamer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode GFXPIPE_3D
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Bh 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 2h — Name: DWORD_COUNT_n **[Default]**
1..2	63:48	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC		
	47:13	**Dx9 Constant Buffer Pool Base Address** Format: GraphicsAddress[47:13]Dx9_Constant_Buffer_Pool This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by an 3DSTATE_CONSTANT_BUFFER_* command when HW generate Dx9 Constant Buffers are enabled. Indirect data accessed at this address and beyond will appear to be 0. Setting this field to 0 will cause this range check to be ignored.
	12:11	**Reserved** Format: MBZ
	10	**Dx9 Constant Buffer Pool Enable** Format: Enable When this bit is set it enables HW Dx9 constants buffers. When this bit is cleared it disables HW Dx9 constant buffers, the local bits for the constant buffers are cleared and the buffers will not be save or restored as part of context.
	9:7	**Reserved** Format: MBZ
	6:0	**Surface Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 2 is not programmable and is always zero.
3	31:13	**Dx9 Constant Buffer Pool Buffer Size** Format: U19 This field specifies the size of the buffer in 8K pages. Any access which straddle or go past the end of the buffer will return 0. Note that BufferSize=0 indicates that there is no valid data in the buffer. **Restriction** Restriction : Programming size of zero is illegal in the case that the pool is enabled.
	12:0	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTB_PS

3DSTATE_DX9_CONSTANTB_PS

Source: RenderCS

Length Bias: 2

This command sets a DX9 constant Boolean register for PS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTB_PS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Eh 3DSTATE_DX9_CONSTANTB_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name 0h **[Default]** 0h-10h Excludes DWord (0,1)
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTB_PS

	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:4	**Reserved** Format: MBZ
	3:0	**Constant Register Index** Format: U4 This field specifies the index of 1st boolean to be updated.
2..n	31:0	**Entry** Format: DX9_CONSTANTB_ENTRY The nth boolean to be updated.

3DSTATE_DX9_CONSTANTB_VS

3DSTATE_DX9_CONSTANTB_VS

Source: RenderCS

Length Bias: 2

This command sets a DX9 constant Boolean register for PS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTB_VS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Dh 3DSTATE_DX9_CONSTANTB_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 0h — Name: **[Default]** Value: 0h-10h — Name: Excludes DWord (0,1)
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTB_VS

	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:4	**Reserved** Format: MBZ
	3:0	**Constant Register Index** Format: U4 This field specifies the index of 1st boolean to be updated.
2..n	31:0	**Entry** Format: DX9_CONSTANTB_ENTRY The nth boolean to be updated.

3DSTATE_DX9_CONSTANTF_PS

3DSTATE_DX9_CONSTANTF_PS

Source: RenderCS

Length Bias: 2

This command sets one or more DX9 constant float registers for PS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTF_PS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Ah 3DSTATE_DX9_CONSTANTF_PS Format: OpCode
	15:11	**Reserved** Format: MBZ
	10:0	**DWord Length** Format: =n Total Length - 2 Value / Name 1h: Excludes DWord (0,1) **[Default]** 1h-400h: multiples of 4
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTF_PS		
	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:8	**Reserved** Format: MBZ
	7:0	**Constant Register Index** Format: U8 This field specifies the index of 1st 4 component float to be updated.
2..n	127:0	**Entry** Format: DX9_CONSTANTF_ENTRY The four components of the nth float to be updated.

3DSTATE_DX9_CONSTANTF_VS

3DSTATE_DX9_CONSTANTF_VS

Source: RenderCS

Length Bias: 2

This command sets one or more DX9 constant float registers for VS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTF_VS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 39h 3DSTATE_DX9_CONSTANTF_VS Format: OpCode
	15:11	**Reserved** Format: MBZ
	10:0	**DWord Length** Format: =n Total Length - 2 Value / Name 1h: Excludes DWord (0,1) **[Default]** 1h-400h: multiples of 4
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTF_VS		
	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:8	**Reserved** Format: MBZ
	7:0	**Constant Register Index** Format: U8 This field specifies the index of 1st 4 component float to be updated.
2..n	127:0	**Entry** Format: DX9_CONSTANTF_ENTRY The four components of the nth float to be updated.

3DSTATE_DX9_CONSTANTI_PS

3DSTATE_DX9_CONSTANTI_PS

Source: RenderCS

Length Bias: 2

This command sets one or more DX9 constant integer registers for PS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTI_PS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Ch 3DSTATE_DX9_CONSTANTI_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 1h / **[Default]** / Excludes DWord (0,1) 0h-80h / multiples of 4 /
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTI_PS

	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:5	**Reserved** Format: MBZ
	4:0	**Constant Register Index** Format: U5 This field specifies the index of 1st 4 component integer to be updated.
2..n	127:0	**Entry** Format: DX9_CONSTANTI_ENTRY The four components of the nth float to be updated.

3DSTATE_DX9_CONSTANTI_VS

3DSTATE_DX9_CONSTANTI_VS

Source: RenderCS

Length Bias: 2

This command sets one or more DX9 constant integer registers for PS.

Programming Notes

- The 3DSTATE_DX9_CONSTANTI_VS is a variable length command.
- Programming this command in batch buffer requires that all float, integer and boolean constants initialized prior to any commands or events that cause the constants to be written to memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Bh 3DSTATE_DX9_CONSTANTI_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 0h / **[Default]** / Excludes DWord (0,1) 0h-80h / multiples of 4 /
1	31:16	**Reserved** Format: MBZ

3DSTATE_DX9_CONSTANTI_VS

	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:5	**Reserved** Format: MBZ
	4:0	**Constant Register Index** Format: U5 This field specifies the index of 1st 4 component integer to be updated.
2..n	127:0	**Entry** Format: DX9_CONSTANTI_ENTRY The four components of the nth float to be updated.

3DSTATE_DX9_GENERATE_ACTIVE_PS

3DSTATE_DX9_GENERATE_ACTIVE_PS

Source: RenderCS

Length Bias: 2

The 3DSTATE_DX9_GENERATE_ACTIVE_PS command is used to generate fixed functions' DX9 Constant Buffer. A DX9 Constant register is made active by writing it out to the constant buffer.

Programming Notes

Restriction:
The global and local buffers are not initialized after reset. Any data written without being initialized will be undefined. DX9 constant buffers are written due to context save/restore or the Generate Active Command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 42h 3DSTATE_DX9_GENERATE_ACTIVE_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:24	**Reserved** Format: MBZ
	23:13	**Pointer to PS Constant Buffer** Format: ConstantBufferOffset[23:13]BINDING_TABLE_STATE* Specifies the 8KB aligned address offset of the PS function's Dx9 constant buffer. This offset is relative to the DX9 Constant buffer Base Address.

3DSTATE_DX9_GENERATE_ACTIVE_PS

12 DX9 Enable

Format:	Enable
Format:	U1

When this bit is set, the Resource Streamer will generate the PS constant buffer according to the DX9 rules:

1. Valid local register are made active.
2. Global register becomes active, unless the corresponding local register is valid.
3. Local register valids are reset.

When this bit is cleared, the Resource Streamer will generate the PS constant buffer according to the DX8 rules:

1. Global register become active.
2. Local register valids are reset.

Programming Notes
In DX8 mode software will set all constants as globals, even ones locally defined within a shader.

11 Clamp Enable

Format:	Enable
Format:	U1

When this bit is set, the Resource Streamer will generate the PS constant buffer with the global values clamped to [-1,1]. When this bit is cleared, the Resource Streamer will generate the PS constant buffer without the global value clamped.

Programming Notes
The clamping affects the values written out to the constant buffer and values read by the Gather engine for push constant buffer generation.

10 Local Valid Bit Reset Enable

Format:	U1

Value	Name	Description
0	Disable	Disable resetting of the local valid bits on the generate active command
1	Enable	Enable resetting of the local valid bits on the generate active command

9 Buffer Write Disable

Format:	U1

When this bit is set, the Resource Streamer will not write the values of the on-die registers to the constant buffer. The internal handshaking will take place as if it did.

Value	Name	Description
0	Enable Writes	On-die register values are written out to the constant buffer.
1	Disable Writes	On-die registers are not written out.

3DSTATE_DX9_GENERATE_ACTIVE_PS		
	8	**Disable Produce to Command Stream** Format: MBZ When set this bit will disable the produce to the command stream for the generation of a constant buffer. This bit must only be set by HW during a context save and restore. Otherwise it will cause the render command stream to hang waiting for a produce. During context save and restore command stream is not parsing the same command buffer.
	7:0	**Reserved** Format: MBZ

3DSTATE_DX9_GENERATE_ACTIVE_VS

3DSTATE_DX9_GENERATE_ACTIVE_VS

Source: RenderCS

Length Bias: 2

The 3DSTATE_DX9_GENERATE_ACTIVE_VS command is used to generate fixed functions' DX9 Constant Buffer. A DX9 Constant register is made active by writing it out to the constant buffer.

Programming Restriction:The global and local buffers are not initialized after reset. Any data written without being initialized will be undefined. DX9 constant buffers are written due to context save/restore or the Generate Active Command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 41h 3DSTATE_DX9_GENERATE_ACTIVE_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:24	**Reserved** Format: MBZ
	23:13	**Pointer to VS Constant Buffer** Format: ConstantBufferOffset[23:13] Specifies the 8KB aligned address offset of the VS function's Dx9 constant buffer. This offset is relative to the DX9 Constant buffer Base Address.

3DSTATE_DX9_GENERATE_ACTIVE_VS

12 DX9 Enable

Format:	Enable

When this bit is set, the Resource Streamer will generate the VS constant buffer according to the DX9 rules:

1. Valid local register are made active.
2. Global register becomes active, unless the corresponding local register is valid.
3. Local register valids are reset.

When this bit is cleared, the Resource Streamer will generate the VS constant buffer according to the DX8 rules:

1. Global register become active.
2. Local register valids are reset.

Programming Notes
In DX8 mode software will set all constants as globals, even ones locally defined within a shader.

11 Clamp Enable

Format:	Enable

When this bit is set, the Resource Streamer will generate the VS constant buffer with the global values clamped to [-1,1]. When this bit is cleared, the Resource Streamer will generate the VS constant buffer without the global value clamped.

Programming Notes
The clamping affects the values written out to the constant buffer and values read by the Gather engine for push constant buffer generation.

10 Local Valid Bit Reset Enable

Format:	Enable

Value	Name	Description
0	Disable	Disable resetting of the local valid bits on the generate active command
1	Enable	Enable resetting of the local valid bits on the generate active command

9 Buffer Write Disable

Format:	Disable

When this bit is set, the Resource Streamer will not write the values of the on-die registers to the constant buffer. The internal handshaking will take place as if it did.

Value	Name	Description
0	Enable Writes	On-die register values are written out to the constant buffer.
1	Disable Writes	On-die registers are not written out.

3DSTATE_DX9_GENERATE_ACTIVE_VS		
	8	**Disable Produce to Command Stream** Format: MBZ When set this bit will disable the produce to the command stream for the generation of a constant buffer. This bit must only be set by HW during a context save and restore. Otherwise it will cause the render command stream to hang waiting for a produce. During context save and restore command stream is not parsing the same command buffer.
	7:0	**Reserved** Format: MBZ

3DSTATE_DX9_LOCAL_VALID_PS

3DSTATE_DX9_LOCAL_VALID_PS

Source: RenderCS

Length Bias: 2

This command sets the local valid bits for the DX9 Constant Buffer

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 40h 3DSTATE_DX9_LOCAL_VALID_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 9h Excludes Dword (0,1) Format: =n Total Length - 2
1..8	31:0	**Local ConstantF Valid Bits** Format: U32 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.
9	31:0	**Local ConstantI Valid Bits** Format: U32 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.
10	31:16	**Reserved** Format: MBZ
	15:0	**Local ConstantB Valid Bits** Format: U16 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.

3DSTATE_DX9_LOCAL_VALID_VS

3DSTATE_DX9_LOCAL_VALID_VS

Source: RenderCS

Length Bias: 2

This command sets the local valid bits for the DX9 Constant Buffer

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 3Fh 3DSTATE_DX9_LOCAL_VALID_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n Total Length - 2
1..8	31:0	**Local ConstantF Valid Bits** Format: U32 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.
9	31:0	**Local ConstantI Valid Bits** Format: U32 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.
10	31:16	**Reserved** Format: MBZ
	15:0	**Local ConstantB Valid Bits** Format: U16 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.

3DSTATE_GATHER_CONSTANT_DS

3DSTATE_GATHER_CONSTANT_DS

Source: RenderCS

Length Bias: 2

This command uses the constant buffer binding table entries to reference constant buffer surface states for the DS unit. The constant data in these is gathered and packed according to a gather table contained in this command.

Programming Notes

The HW generated binding table must be enabled to use this command.

The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.

When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.

If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.

The 3DSTATE_GATHER_CONSTANT_* command is not committed to the resource streamer engine until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command. For example, the 3DSTATE_GATHER_CONSTANT_VS command will not actually generate a buffer in memory until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the resource streamer.

The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory:
•Resource Streamer Flush = 1
•3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h.
•3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_GATHER_CONSTANT_DS

26:24 3D Command Opcode

Default Value:	0h 3DSTATE_PIPELINED
Format:	OpCode

23:16 3D Command Sub Opcode

Default Value:	37h 3DSTATE_GATHER_CONSTANT_DS
Format:	OpCode

15:8 Reserved

Format:	MBZ

7:0 DWord Length

Format:	=n

Total Length - 2

Value	Name	Description
1	DWORD_COUNT_n **[Default]**	excludes DWords 0,1
[1,128]	Range	1-128 Entries

1 31:16 Constant Buffer Valid

Format:	U16

This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.

15:12 Constant Buffer Binding Table Block

Format:	U4

This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.

11:2 Reserved

Format:	MBZ

1 Update Gather Table Only

Format:	U1

When this bit is set, the Resource Streamer will update the on chip buffer of Gather Index values. Any following Binding Table Pointer will not commit this command to the Gather Engine.

Value	Name	Description
0	Commit Gather	Update Gather Table and Commit Gather on next BTP.
1	Non-Commit Gather	Update Gather Table Only.

0 Reserved

Format:	MBZ

2 31:23 Reserved

Format:	MBZ

3DSTATE_GATHER_CONSTANT_DS

	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool **Programming Notes** SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Format: MBZ
	3	**On-Die Table** Format: U1 This bit controls whether the on-die table is loaded or read. **Value** / **Name** / **Description** 0h / Load / Load the on-die table w/ table contained in this command and do the gather using the table in this command. 1h / Read / Read the on-die table and use its contents for doing the gather command. **Programming Notes** The HW will overwrite DW1 and DW2 of the previous GATHER with the current DW1 and DW2. If using the same gather tables, then the valid bits must be still in sync with the indexes used in the previous GATHER table. The Dword Length field must be set to 1h when this field is enabled.
	2:0	**Reserved** Format: MBZ
3..n	31:16	**Entry_1** Format: GATHER_CONSTANT_ENTRY
	15:0	**Entry_0** Format: GATHER_CONSTANT_ENTRY

3DSTATE_GATHER_CONSTANT_GS

3DSTATE_GATHER_CONSTANT_GS

Source: RenderCS

Length Bias: 2

This command uses the constant buffer binding table entries to reference constant buffer surface states for GS unit. The constant data in these is gathered and packed according to a gather table contained in this command.

Programming Notes

The HW generated binding table must be enabled to use this command.

The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.

When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.

If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.

The 3DSTATE_GATHER_CONSTANT_* command is not committed to the resource streamer engine until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command. For example, the 3DSTATE_GATHER_CONSTANT_VS command will not actually generate a buffer in memory until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the resource streamer.

The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory:
•Resource Streamer Flush = 1
•3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h.
•3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_GATHER_CONSTANT_GS

26:24 3D Command Opcode

Default Value:	0h 3DSTATE_PIPELINED
Format:	OpCode

23:16 3D Command Sub Opcode

Default Value:	35h 3DSTATE_GATHER_CONSTANT_GS
Format:	OpCode

15:8 Reserved

Format:	MBZ

7:0 DWord Length

Format:	=n

Total Length - 2

Value	Name	Description
1	DWORD_COUNT_n **[Default]**	excludes DWords 0,1
[1,128]	Range	1-128 Entries

1 31:16 Constant Buffer Valid

Format:	U16

This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.

15:12 Constant Buffer Binding Table Block

Format:	U4

This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.

11:2 Reserved

Format:	MBZ

1 Update Gather Table Only

Format:	U1

When this bit is set, the Resource Streamer will update the on chip buffer of Gather Index values. Any following Binding Table Pointer will not commit this command to the Gather Engine.

Value	Name	Description
0	Commit Gather	Update Gather Table and Commit Gather on next BTP.
1	Non-Commit Gather	Update Gather Table Only.

0 Reserved

Format:	MBZ

3DSTATE_GATHER_CONSTANT_GS

2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool. **Programming Notes** SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Format: MBZ
	3	**On-Die Table** Format: U1 This bit controls whether the on-die table is loaded or read. **Value** / **Name** / **Description** 0h / Load / Load the on-die table w/ table contained in this command and do the gather using the table in this command. 1h / Read / Read the on-die table and use its contents for doing the gather command. **Programming Notes** The HW will overwrite DW1 and DW2 of the previous GATHER with the current DW1 and DW2. If using the same gather tables, then the valid bits must be still in sync with the indexes used in the previous GATHER table. The Dword Length field must be set to 1h when this field is enabled.
	2:0	**Reserved** Format: MBZ
3..n	31:16	**Entry_1** Format: GATHER_CONSTANT_ENTRY
	15:0	**Entry_0** Format: GATHER_CONSTANT_ENTRY

3DSTATE_GATHER_CONSTANT_HS

3DSTATE_GATHER_CONSTANT_HS

Source: RenderCS

Length Bias: 2

This command uses the constant buffer binding table entries to reference constant buffer surface states for HS unit. The constant data in these is gathered and packed according to a gather table contained in this command.

Programming Notes

The HW generated binding table must be enabled to use this command.

The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.

When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.

If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.

The 3DSTATE_GATHER_CONSTANT_* command is not committed to the resource streamer engine until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command. For example, the 3DSTATE_GATHER_CONSTANT_VS command will not actually generate a buffer in memory until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the resource streamer.

The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory:
•Resource Streamer Flush = 1
•3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h.
•3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_GATHER_CONSTANT_HS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 36h 3DSTATE_GATHER_CONSTANT_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,128] / Range / 1-128 Entries
1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1	**Update Gather Table Only** Format: U1 When this bit is set, the Resource Streamer will update the on chip buffer of Gather Index values. Any following Binding Table Pointer will not commit this command to the Gather Engine. **Value** / **Name** / **Description** 0 / Commit Gather / Update Gather Table and Commit Gather on next BTP. 1 / Non-Commit Gather / Update Gather Table Only.
	0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ

3DSTATE_GATHER_CONSTANT_HS

	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool. **Programming Notes** SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Format: MBZ
	3	**On-Die Table** Format: U1 This bit controls whether the on-die table is loaded or read.

Value	Name	Description
0h	Load	Load the on-die table w/ table contained in this command and do the gather using the table in this command.
1h	Read	Read the on-die table and use its contents for doing the gather command.

Programming Notes
The HW will overwrite DW1 and DW2 of the previous GATHER with the current DW1 and DW2. If using the same gather tables, then the valid bits must be still in sync with the indexes used in the previous GATHER table.
The Dword Length field must be set to 1h when this field is enabled.

	2:0	**Reserved** Format: MBZ
3..n	31:16	**Entry_1** Format: GATHER_CONSTANT_ENTRY
	15:0	**Entry_0** Format: GATHER_CONSTANT_ENTRY

3DSTATE_GATHER_CONSTANT_PS

3DSTATE_GATHER_CONSTANT_PS

Source: RenderCS

Length Bias: 2

This command uses the constant buffer binding table entries to reference constant buffer surface states for PS unit. The constant data in these is gathered and packed according to a gather table contained in this command.

Programming Notes

The HW generated binding table must be enabled to use this command.

The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.

When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.

If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.

The 3DSTATE_GATHER_CONSTANT_* command is not committed to the resource streamer engine until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command. For example, the 3DSTATE_GATHER_CONSTANT_VS command will not actually generate a buffer in memory until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the resource streamer.

The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory:
•Resource Streamer Flush = 1
•3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h.
•3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED

3DSTATE_GATHER_CONSTANT_PS

		Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 38h 3DSTATE_GATHER_CONSTANT_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2

Value	Name	Description
1	DWORD_COUNT_n **[Default]**	excludes DWords 0,1
[1,128]	Range	1-128 Entries

1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1	**Update Gather Table Only** Format: U1 When this bit is set, the Resource Streamer will update the on chip buffer of Gather Index values. Any following Binding Table Pointer will not commit this command to the Gather Engine.

Value	Name	Description
0	Commit Gather	Update Gather Table and Commit Gather on next BTP.
1	Non-Commit Gather	Update Gather Table Only.

	0	**DX9 On-Die Register Read Enable** Format: Enable When this bit is set, the Resource Streamer will not read out to the constant buffer, but instead it will read from the on-die registers. The internal handshaking will take place as if it did perform the read from the constant buffer. The **Constant Buffer Offset** is the offset into the on-die register and the **Binding Table Index Offset** will cause it to be a 4KB offset if the value is '1'. If the **Binding Table Index Offset** is '0' then the engine will read from the lower 4KB of the

3DSTATE_GATHER_CONSTANT_PS

		constant buffer. This bit only has effect if the Dx9 enable is set.
2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool. **Programming Notes** SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Constant Buffer Dx9 Enable** Format: Enable When this bit is set it indicates that the constant buffer is a HW generated Dx9 constant buffer. The resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization. Additionally the Dx9 constant buffers are a single buffer but larger than 4KB. Internally the HW will treat the DX9 buffer as 2 constant buffers. When this bit is enable only the 1st constant buffer valid bit is set. The 2nd constant buffer surface pointer will automatically be the 1st pointer + 4KB.
	3	**On-Die Table** Format: U1 This bit controls whether the on-die table is loaded or read. **Value** / **Name** / **Description** 0h / Load / Load the on-die table w/ table contained in this command and do the gather using the table in this command. 1h / Read / Read the on-die table and use its contents for doing the gather command. **Programming Notes** The HW will overwrite DW1 and DW2 of the previous GATHER with the current DW1 and DW2. If using the same gather tables, then the valid bits must be still in sync with the indexes used in the previous GATHER table. The Dword Length field must be set to 1h when this field is enabled.
	2:0	**Reserved** Format: MBZ
3..n	31:16	**Entry_1**

3DSTATE_GATHER_CONSTANT_PS			
		Format:	GATHER_CONSTANT_ENTRY
	15:0	**Entry_0**	
		Format:	GATHER_CONSTANT_ENTRY

3DSTATE_GATHER_CONSTANT_VS

3DSTATE_GATHER_CONSTANT_VS

Source: RenderCS

Length Bias: 2

This command uses the constant buffer binding table entries to reference constant buffer surface states for VS unit. The constant data in these is gathered and packed according to a gather table contained in this command.

Programming Notes

The HW generated binding table must be enabled to use this command.

The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.

When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.

If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.

The 3DSTATE_GATHER_CONSTANT_* command is not committed to the resource streamer engine until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command. For example, the 3DSTATE_GATHER_CONSTANT_VS command will not actually generate a buffer in memory until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the resource streamer.

Workaround

The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory: •Resource Streamer Flush = 1 •3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h. •3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_GATHER_CONSTANT_VS

26:24 3D Command Opcode

Default Value:	0h 3DSTATE_PIPELINED
Format:	OpCode

23:16 3D Command Sub Opcode

Default Value:	34h 3DSTATE_GATHER_CONSTANT_VS
Format:	OpCode

15:8 Reserved

Format:	MBZ

7:0 DWord Length

Format:	=n

Total Length - 2

Value	Name	Description
0	DWORD_COUNT_n **[Default]**	excludes DWords 0,1
[1,128]	Range	1-128 Entries

1 31:16 Constant Buffer Valid

Format:	U16

This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.

15:12 Constant Buffer Binding Table Block

Format:	U4

This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.

11:2 Reserved

Format:	MBZ

1 Update Gather Table Only

Format:	U1

When this bit is set, the Resource Streamer will update the on chip buffer of Gather Index values. Any following Binding Table Pointer will not commit this command to the Gather Engine.

Value	Name	Description
0	Commit Gather	Update Gather Table and Commit Gather on next BTP.
1	Non-Commit Gather	Update Gather Table Only.

3DSTATE_GATHER_CONSTANT_VS

0 — DX9 On-Die Register Read Enable

Format:	Enable

When this bit is set, the Resource Streamer will not read out to the constant buffer, but instead it will read from the on-die registers. The internal handshaking will take place as if it did perform the read from the constant buffer. The **Constant Buffer Offset** is the offset into the on-die register and the **Binding Table Index Offset** will cause it to be a 4KB offset if the value is '1'. If the **Binding Table Index Offset** is '0' then the engine will read from the lower 4KB of the constant buffer. This bit only has effect if the Dx9 enable is set.

2 — 31:23 — Reserved

Format:	MBZ

22:6 — Gather Buffer Offset

Format:	GatherBufferOffset[22:6]

This field specifies the offset of the gather buffer within the Gather Pool.

Programming Notes
SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.

5 — Constant Buffer Dx9 Generate Stall

Format:	Enable

When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.

4 — Constant Buffer Dx9 Enable

Format:	Enable

When this bit is set it indicates that the constant buffer is a HW generated Dx9 constant buffer. The resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization. Additionally the Dx9 constant buffers are a single buffer but larger than 4KB. Internally the HW will treat the DX9 buffer as 2 constant buffers. When this bit is enable only the 1st constant buffer valid bit is set. The 2nd constant buffer surface pointer will automatically be the 1st pointer + 4KB.

3 — On-Die Table

Format:	U1

This bit controls whether the on-die table is loaded or read.

Value	Name	Description
0h	Load	Load the on-die table w/ table contained in this command and do the gather using the table in this command.
1h	Read	Read the on-die table and use its contents for doing the gather command.

Programming Notes
The HW will overwrite DW1 and DW2 of the previous GATHER with the current DW1 and DW2. If using the same gather tables, then the valid bits must be still in sync with the indexes used in

3DSTATE_GATHER_CONSTANT_VS		
		the previous GATHER table. The Dword Length field must be set to 1h when this field is enabled.
	2:0	**Reserved** Format: MBZ
3..n	31:16	**Entry_1** Format: GATHER_CONSTANT_ENTRY
	15:0	**Entry_0** Format: GATHER_CONSTANT_ENTRY

3DSTATE_GATHER_POOL_ALLOC

3DSTATE_GATHER_POOL_ALLOC

Source: RenderCS

Length Bias: 2

This command sets up the Gather Pool for Gather Buffers.

Programming Notes

If the Gather Constant Pool is enabled and RS is enabled, then for each 3DSTATE_CONSTANT_* command there must be a corresponding 3DSTATE_GATHER_CONSTANT_* command. If gather pool is enabled, then Buffer 1 of the 3DSTATE_CONSTANT command address will be an offset from the Gather Pool Base Address.

The gather constants can only be enabled if the binding table generator is also enabled. This command must only be programmed when resource streamer is enabled to parse commands within a batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ah 3DSTATE_GATHER_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 2h — Name: DWORD_COUNT_n **[Default]**
1..2	63:12	**Gather Pool Base Address** Format: GraphicsAddress63-12 GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

3DSTATE_GATHER_POOL_ALLOC

	11	**Gather Pool Enable** Format: Enable When this bit is set it enables HW gathering of push constants. When this bit is cleared it disables HW gathering of push constants.
	10:7	**Reserved** Format: MBZ
	6:0	**Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 2 is not programmable and is always zero.
3	31:12	**Gather Pool Buffer Size** Format: U20 This field specifies the size of the buffer in 4K pages. Any access which straddle or go past the end of the buffer will return undefined data. Note that BufferSize=0 indicates that there is no valid data in the buffer. **Value** / **Name** [0,1048575] **Restriction** Restriction : Programming size of zero is illegal in the case that the pool is enabled.
	11:0	**Reserved** Format: MBZ

3DSTATE_GS

3DSTATE_GS

Source: RenderCS

Length Bias: 2

Controls the GS stage hardware.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 11h 3DSTATE_GS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 8h — Name: Excludes DWord (0,1) **[Default]**
1..2	63:6	**Kernel Start Pointer** Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location (1st GEN4 core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.
	5:0	**Reserved** Format: MBZ
3	31	**Single Program Flow** Specifies the initial condition of the kernel program as either a single program flow (SIMDnxm with m = 1) or as multiple program flows (SIMDnxm with m > 1). See CR0 description in ISA Execution Environment.

3DSTATE_GS

Value	Name	Description
0h	Disable	Single Program Flow disabled
1h	Enable	Single Program Flow enabled

30 **Vector Mask Enable**

Format:	Enable Enumerated Type

Upon subsequent GS thread dispatches, this bit is loaded into the EU's Vector Mask Enable (VME, cr0.0[3]) thread state. Refer to EU documentation for the definition and use of VME state.

Value	Name	Description
0h	Dmask	The EU will use the Dispatch Mask (supplied by the GS stage) for instruction execution.
1h	Vmask	The EU will use the Vector Mask (derived from Dispatch Mask) for instruction execution.

Programming Notes
Under normal conditions SW shall specify DMask, as the GS stage will provide a Dispatch Mask appropriate to SIMD4x2 or SIMD8 thread execution (as a function of dispatch mode). E.g., for SIMD4x2 execution, the GS stage will generate a Dispatch Mask that is equal to what the EU would use as the Vector Mask. For SIMD8 execution there is no known usage model for use of Vector Mask (as there is for PS shaders).

29:27 **Sampler Count**

Format:	U3

Specifies how many samplers (in multiples of 4) the geometry shader kernel uses. Used only for prefetching the associated sampler state entries.

Value	Name	Description
0h	No Samplers	No Samplers used
1h	1-4 Samplers	Between 1 and 4 samplers used
2h	5-8 Samplers	Between 5 and 8 samplers used
3h	9-12 Samplers	Between 9 and 12 samplers used
4h	13-16 Samplers	Between 13 and 16 samplers used
5h-7h	Reserved	

26 **Reserved**

25:18 **Binding Table Entry Count**

Format:	U8

When **HW Generated Binding Table** is disabled:

Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.

Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.

When **HW Generated Binding Table** bit is enabled:

3DSTATE_GS

This field indicates which cache lines (512bit units - 32 Binding Table Entry section) should be fetched. Each bit in this field corresponds to a cache line. Only the 1st 4 non-zero Binding Table entries of each 32 Binding Table entry section prefetched will have its surface state prefetched.

Programming Notes
When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be generated at JIT time.

17 Thread Dispatch Priority

Specifies the priority of the thread for dispatch.

Value	Name	Description
0h	Normal	Normal thread dispatch priority
1h	High	High thread dispatch priority

16 Floating Point Mode

Specifies the initial floating point mode used by the dispatched thread.

Value	Name	Description
0h	IEEE-754	Use IEEE-754 Rules
1h	Alternate	Use alternate rules

15:14 Reserved

Format:	MBZ

13 Illegal Opcode Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions and ISA Execution Environment*.

12 Accesses UAV

Format:	Enable

This field must be set when GS has a UAV access.

Programming Notes
This field must not be set when GS Function Enable is disabled.

11 Mask Stack Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[11]. See *Exceptions and ISA Execution Environment.*

10:8 Reserved

Format:	MBZ

7 Software Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions and ISA Execution Environment.*

3DSTATE_GS

	6	**Reserved** Format: MBZ
	5:0	**Expected Vertex Count** Format: U6 Specifies the number of vertices per input object expected by the GS thread. Input topologies not matching this expect value are discarded. Note that **DiscardAdjacency** is also considered (e.g., if the value programmed is 3 and DiscardAdjacency is set, TRILIST_ADJ and TRISTRIP_ADJ topologies are not discarded as they will pass 3 vertices/object to the GS threads). **Value** [1,32] **Name**
4..5	63:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[63:10]ScratchSpace Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if VS Function Enable is DISABLED.
	9:4	**Reserved** Format: MBZ
	3:0	**Per-Thread Scratch Space** Format: U4 power of 2 Bytes over 1K Bytes Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. **Value** [0,11] **Name** **Description** indicating [1K Bytes, 2M Bytes]
6	31	**Reserved** Format: MBZ
	30:29	**Dispatch GRF Start Register For URB Data [5:4]** Format: U2 Specifies bit [5:4] of the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. The **Dispatch GRF Start Register For URB Data [3:0]** field is used to specify bits [3:0] of the starting GRF register number.

3DSTATE_GS

28:23 Output Vertex Size

Format:	U6

[0,63] indicating [1,64] 16B units

Specifies the size of each vertex stored in the GS output entry (following any Control Header data) as a number of 128-bit units (minus one).

Programming Notes

Programming Restrictions: The vertex size must be programmed as a multiple of 32B units with the following exception: Rendering is disabled (as per SOL stage state) and the vertex size output by the GS thread is 16B.

If rendering is enabled (as per SOL state) the vertex size must be programmed as a multiple of 32B units. In other words, the only time software can program a vertex size with an odd number of 16B units is when rendering is disabled.

22:17 Output Topology

Format:	3D_Prim_Topo_Type

This field specifies the topology type (3DPrimType) to be associated with GS-thread output vertices (if any).

16:11 Vertex URB Entry Read Length

Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments.

Programming Notes

Programming Restriction:This field must be a non-zero value if Include Vertex Handles is cleared to zero.

10 Include Vertex Handles

Format:	Boolean

If set, all the input Vertex URB handles are included in the payload. These are referred to as "pull model" URB handles, as the thread will use them to read from the URB.

Programming Notes

Programming Restriction: This field must be set if Vertex URB Entry Read Length is cleared to zero.

9:4 Vertex URB Entry Read Offset

Format:	U6

Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread.

3:0 Dispatch GRF Start Register For URB Data

Format:	U4

3DSTATE_GS

Description
Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.
The Dispatch GRF Start Register for URB Data [5:4] field is used to extend the range of the starting GRF register number to [0,63].

Value	Name	Description
[0,15]		indicating bits [3:0] of the GRF number

Programming Notes
If Include Vertex Handles is enabled (pull or hybrid handles case), then For simd4x2: For DUAL_OBJECT dispatch mode this field should be: (((2*numVerticesPerObject) + 8 - 1)/8) + 1 For SINGLE and DUAL_INSTANCE dispatch modes this field should be: ((numVerticesPerObject +8 - 1)/8) + 1 If Include Primitive ID is set, then add 1 to the value obtained by using the above
If Include Vertex Handles is enabled (pull or hybrid handles case), then For simd8: For InstanceCount == 1: numVerticesPerObject 2 For InstanceCount > 1: ((numVerticesPerObject 8 - 1)/8) 2 If Include Primitive ID is set, then add 1 to the value obtained by using the above

7 — 31:26 **Reserved**

Format:	MBZ

25:24 **Reserved**

Format:	MBZ

23:20 **Control Data Header Size**

Format:	U4

Specifies the number of 32B units of control data header located at the start of the GS URB entry. The value 0 indicates there is no control data header, and Control Data Format is ignored. Software must ensure that the Control Data Header Size is sufficient to accommodate the maixumum number of vertices output by the GS thread. It is UNDEFINED for a GS thread to report more output vertices than can be accomodated in a non-zero-sized header. (If the header size is zero, by definition neither cut nor StreamID bits are defined.

Value	Name
[0,8]	32B Units

3DSTATE_GS

19:15 Instance Control

Format:	U5-1 #Instances

Specifies the number of instances (minus one) for each input object. To avoid confusion, this document uses the term **"InstanceCount"** to refer to InstanceControl+1, with a range of [1,32]
If **InstanceCount>1**, DUAL_OBJECT mode is invalid. Software will likely want to use DUAL_INSTANCE mode for higher performance, but SINGLE mode is also supported.
When **InstanceCount=1** (one instance per object), software can decide which dispatch mode to use. DUAL_OBJECT mode would likely be the best choice for performance, followed by SINGLE mode. DUAL_INSTANCE mode is not recommended but is supported.

Value	Name	Description
[0,31]		Indicating [1,31] instances

14:13 Default Stream Id

Format:	U2

When the GS is enabled, unless the GS output entry contains StreamID bits in the control header, this field specifies the default StreamID associated with any GS-thread output vertices. When the GS is disabled, StreamID will be output as 0.

12:11 Dispatch Mode

Format:	U2

This field specifies how the GS unit dispatches multiple instances and/or multiple objects.

Value	Name	Description	Programming Notes
0h	Single	Each thread shades a single instance of one object.	
1h	Dual Instance	Each thread shades possibly two instances of one object. If the InstanceCount is odd, a trailing dispatch of only one instance will be made for each object received. Not recommended if InstanceCount = 1, assuming a kernel optimized for SINGLE or DUAL_OBJECT dispatch would outperform a kernel compiled for DUAL_INSTANCE but only passed one instance.	
2h	Dual Object	Each thread shades one instance of possibly two objects. The GS unit attempt to pair objects together into one dispatch, but under some circumstances only one object may be dispatched (as controlled by the DispatchMask generated by the GS unit). Not valid for objects with more than 16 vertices per object. Not valid if InstanceCount > 1 (more than one instance per object).	

3DSTATE_GS

3h	SIMD8	Each thread shades up to 8 different objects or (if InstanceCount >1) 8 instances of a single object.	The driver must send pipe control with a cs stall after a 3dstate_gs state change and the Dispatch Mode is simd8 and the number of handles allocated to gs is less than 16.

Programming Notes
The GS must be allocated at least two URB handles or behavior is UNDEFINED for Dual Instance or Dual Object mode.

10 **Statistics Enable**

Format:	Enable

This bit controls whether GS-unit-specific statistics register(s) can be incremented.

Value	Name	Description
0h	Disable	GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT cannot increment
1h	Enable	GS_INVOCATIONS_COUNT and GS_PRIMITIVES_COUNT can increment

9:5 **Invocations Increment Value**

Format:	U5

Specifies how much to increment the GS_INVOCATIONS_COUNT for each instance of each object. This control is provided to allow software to process multiple instances (from an API POV) in a single kernel invocation.

In SINGLE dispatch mode, the counter will increment by this value for each dispatch (as it's only one instance of one object).

In DUAL_INSTANCE mode, the counter will be incremented by the value if only one instance is included in the dispatch (i.e., the last odd instance), otherwise the counter will be incremented by twice this value.

In DUAL_OBJECT dispatch mode, the counter will be incremented by the value if only one object is included in the dispatch (i.e., a forced dispatch of one object), otherwise the counter will be incremented by twice this value.

Value	Name	Description
[0,31]		indicating an increment of [1,32]

4 **Include Primitive ID**

Format:	Boolean

If set, R1 of the payload is written with Primitive ID value(s).

If clear, these Primitive ID values are not included in the payload R1.

3 **Hint**

Format:	U1

This state bit is simply passed in GS thread payloads for use by the GS kernel - it has no other impact on hardware operation.

3DSTATE_GS

2 Reorder Mode

This bit controls how vertices of triangle objects resulting from TRISTRIP[_ADJ][_REV] topologies are [re]ordered when passed in the GS thread payload See Object Vertex Ordering table (below).

Value	Name	Description
0h	LEADING	Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the leading (first) vertices are in consecutive order starting at v0. A similar reordering is performed on alternating triangles in a TRISTRIP_REV.
1h	TRAILING	Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the trailing (last) vertices are in consecutive order starting at v2. A similar reordering is performed on alternating triangles in a TRISTRIP_REV.

Programming Notes

Workaround: reorder mode must be set to REORDER_LEADING and reordering must be done in the Geometry shader.

1 Discard Adjacency

Format:	Enable

When set, adjacent vertices will not be passed in the GS payload when objects with adjacency are processed. Instead, only the non-adjacent vertices will be passed in the same fashion as the without-adjacency form of the primitive. Software should set this bit whenever a GS kernel is used that does not expect adjacent vertices. This allows both with-adjacency/without-adjacency variants of the primitive to be submitted to the pipeline (via 3DPRIMITIVE) - the GS unit will silently discard any adjacent vertices and present the GS thread with only the internal object. When clear, adjacent vertices will be passed to the GS thread, as dictated by the incoming primitive type. Software should only clear this bit when a GS kernel is used that does expect adjacent vertices. E.g., if the GS kernel is compiled to expect a TRIANGLE_ADJ object, software must clear this bit. Software should also clear this bit if the GS kernel expects a POINT or PATCHLIST_n object (which don't have with-adjacency variants).

The only hardware assistance is to allow the submission of a with-adjacency variant of a primitive when operating with a GS kernel that expects the without-adjacency variant of the object. (E.g., when the GS kernel is compiled to expect a TRIANGLE object, software should set this bit just in case a TRILIST_ADJ is submitted to the pipeline.) Note that the GS unit is otherwise not aware of the object type that is expected by the GS kernel. It is up to software to ensure that the submitted primitive type (in 3DPRIMITIVE) is otherwise compatible with the object type expected by the GS kernel. (E.g., if the GS kernel expects a LINE_ADJ object, only LINELIST_ADJ or LINESTRIP_ADJ should be submitted, otherwise the GS kernel will produce unpredictable results.) Also note that it is possible to craft a GS kernel which can accept any object type that's thrown at it by first examining the PrimType passed in the payload and then using this info to correctly interpret the number of vertices passed in the payload.

0 Enable

Format:	Enable

Specifies whether the GS stage is enabled or disabled (pass-through).

3DSTATE_GS

8 / 31 — Control Data Format

Format:	U1

This field specifies the format of the control data header (if any).

Value	Name	Description
0h	CUT	The control data header contains cut bits.
1h	SID	The control data header contains StreamID bits. . Output Topology must be set to POINTLIST, or behavior is UNDEFINED.

30 — Static Output

Format:	Enable

Specifies whether the GS shader outputs a static number of vertices per invocation.
If this bit is clear, the number of vertices output by each GS shader invocation is stored by the GS thread at the very beginning of the output URB entry (see GS URB Entry section below).

29:27 — Reserved

Format:	MBZ

26:16 — Static Output Vertex Count

Format:	U11 Count of object vertices

If GSEnable is set and StaticOutput is set, this field specifies the total number of vertices output each GS shader invocation.
If **GSEnable** is set and StaticOutput is clear (variable GS output), the total number of vertices output by a GS shader invocation is stored by the thread at the very beginning of the output URB entry. This field is then ignored. (See GS URB Entry below).

15:9 — Reserved

Format:	MBZ

8:0 — Maximum Number of Threads

Format:	U9-1 Thread count

Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance.

Value	Name	Description
[7,335]		indicating thread count of [8,336]

9 / 31:28 — Reserved

Format:	MBZ

27 — Reserved

3DSTATE_GS

26:21 Vertex URB Entry Output Read Offset

Format:	U6

Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB by SBE.

Value	Name
[0,63]	

20:16 Vertex URB Entry Output Length

Format:	U5

Specifies the amount of URB data written for each Vertex URB entry, in 256-bit register increments.

Value	Name
[1,16]	

Programming Notes
This length does not include the vertex header.

15:8 User Clip Distance Clip Test Enable Bitmask

Format:	Enable[8]

This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made.
DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

7:0 User Clip Distance Cull Test Enable Bitmask

Format:	Enable[8]

This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip).
DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_HIER_DEPTH_BUFFER

Source: RenderCS

Length Bias: 2

Description

This command sets the surface state of the hierarchical depth buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

Programming Notes

Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set, followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set, followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 07h 3DSTATE_HIER_DEPTH_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Format: =n Total Length - 2 Value: 3h — Name: Excludes Dword (0,1) **[Default]**

3DSTATE_HIER_DEPTH_BUFFER

1	31:25	**Hierarchical Depth Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the hierarchical depth buffer.
	24:23	**Reserved** Format: MBZ
	22	**Reserved** Format: MBZ
	21:17	**Reserved** Format: MBZ
	16:0	**Surface Pitch** Format: U17-1 Pitch in Bytes This field specifies the pitch of the hierarchical depth buffer in (#Bytes - 1). **Value** | **Name** [127, 1FFFFh] | corresponding to [128B, 128KB] also restricted to a multiple of 128B **Programming Notes** Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].
2..3	63:0	**Surface Base Address** Format: GraphicsAddress[63:0]HierarchicalDepthBuffer This field specifies the address of the buffer in Graphics Memory. **Programming Notes** The Hierarchical Depth Buffer can only be mapped to Main Memory (uncached).
4	31:15	**Reserved** Format: MBZ
	14:0	**Surface QPitch** Format: QPitch[16:2] **Description** The interpretation of this field is dependent on **Surface Type** as follows: • SURFTYPE_1D: distance in *pixels* between array slices • SURFTYPE_2D/CUBE: distance in *rows* between array slices • SURFTYPE_3D: distance in *rows* between R-slices **Value** | **Name** | **Description** [4h, 1FFFCh] | | in multiples of 4 (low 2 bits missing)

3DSTATE_HIER_DEPTH_BUFFER		
		Programming Notes This field must be set to an integer multiple of 8 (QPitch[2] MBZ) Software must ensure that this field is set to a value sufficiently large such that the array slices in the surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored in memory.

3DSTATE_HS

3DSTATE_HS

Source: RenderCS

Length Bias: 2

Controls the HS stage hardware.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Bh 3DSTATE_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 7 — Name: Excludes DWord (0,1) **[Default]**
1	31:30	**Reserved** Format: MBZ
	29:27	**Sampler Count** Format: U3 Specifies how many samplers (in multiples of 4) the HS kernels use. Used only for prefetching the associated sampler state entries. Value: 0h — Name: No Samplers — Description: no samplers used Value: 1h — Name: 1-4 Samplers — Description: between 1 and 4 samplers used Value: 2h — Name: 5-8 Samplers — Description: between 5 and 8 samplers used

3DSTATE_HS

	3h	9-12 Samplers	between 9 and 12 samplers used
	4h	13-16 Samplers	between 13 and 16 samplers used
	5h-7h	Reserved	Reserved

26 Reserved

Format:	MBZ

25:18 Binding Table Entry Count

Format:	U8

When HW Generated Binding Table is disabled:
Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.
Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.

Programming Notes
When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be generated at JIT time.

17 Thread Dispatch Priority
Specifies the priority of the thread for dispatch

Value	Name	Description
0h	Normal	Normal Priority
1h	High	High Priority

16 Floating Point Mode
Specifies the initial floating point mode used by the dispatched thread.

Value	Name	Description
0h	IEEE-754	Use IEEE-754 Rules
1h	alternate	Use alternate rules

15:14 Reserved

Format:	MBZ

13 Illegal Opcode Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.

12 Software Exception Enable

Format:	Enable

This bit gets loaded into EU CRO1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.

11:8 Reserved

Format:	MBZ

3DSTATE_HS

	7:0	**Reserved** Format: MBZ
2	31	**Enable** Format: Enable Specifies whether the HS function is enabled or disabled (pass-through). If ENABLED MI_TOPOLOGY_FILTER must be used to silently discard any topologies that the HS kernel is not expecting. E.g., if the HS kernel is expecting PATCHLIST_32 topologies, MI_TOPOLOGY_FILTER must be set to PATCHLIST_32 so only those topologies can reach the enabled HS. **Programming Notes** The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.
	30	**Reserved** Format: MBZ
	29	**Statistics Enable** Format: Enable This bit controls whether HS-unit-specific statistics register(s) will increment (for each patch).
	28:27	**Reserved** Format: MBZ
	26:18	**Reserved** Format: MBZ
	17	**Reserved** Format: MBZ
	16:8	**Maximum Number of Threads** Format: U9-1 Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance. **Value** / **Name** / **Description** [0,335] / / indicating thread count of [1,336]
	7:5	**Reserved** Format: MBZ
	4	**Reserved** Format: MBZ

3DSTATE_HS

	3:0	**Instance Count** Format: U4-1 This field determines the number of threads (minus one) spawned per input patch. If the HS kernel uses a barrier function, software must restrict the **Instance Count** to the number of threads that can be simultaneously active within a subslice. Factors which must be considered includes scratch memory availability. **Value** / **Name** / **Description** [0,15] / / representing [1,16] instances **Programming Notes** Instance Count must be programmed to 0 (1 instance) whenever DispatchMode is programmed to DUAL_PATCH.
3..4	63:6	**Kernel Start Pointer** Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location (1st GEN core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.
	5:0	**Reserved** Format: MBZ
5..6	63:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[63:10] **Value** / **Name** / **Description** [0,31] / / Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.
	9:4	**Reserved** Format: MBZ
	3:0	**Per-Thread Scratch Space** Format: U4 power of 2 Bytes over 1K Bytes Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. **Value** / **Name** / **Description** [0,11] / / Indicating[1K Bytes, 2M Bytes

3DSTATE_HS

7 — 31:29 — Reserved

Format:	MBZ

28 — Dispatch GRF Start Register For URB Data [5]

Format:	U1

Specifies bit [5] of the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload. The **Dispatch GRF Start Register For URB Data [4:0]** field is used to specify bits [4:0] of the starting GRF register number.

27 — Single Program Flow

Format:	Enable

Specifies the initial condition of the kernel program as either a single program flow (SIMDnxm with m = 1) or as multiple program flows (SIMDnxm with m > 1). See CR0 description in *ISA Execution Environment*.

Value	Name	Description
0h	Reserved	
1h	Enable	Single Program Flow Enabled

26 — Vector Mask Enable

Format:	U1 Enumerated Type

Upon subsequent HS thread dispatches, this bit is loaded into the EU's Vector Mask Enable (VME, cr0.0[3]) thread state. Refer to the EU documentation for the definition and use of VME state.

Value	Name	Description
0h	Dmask	The EU will use the Dispatch Mask (supplied by the HS stage) for instruction execution.
1h	Vmask	The EU will use the Vector Mask (derived from the Dispatch Mask) for instruction execution.

Programming Notes
Under normal conditions SW shall specify DMask, as the HS stage will provide a Dispatch Mask appropriate to SIMD4x2 or SIMD8 thread execution (as a function of dispatch mode). E.g., for SIMD4x2 thread execution, the HS state will generate a Dispatch Mask that is equal to what the EU would use as a Vector Mask. For SIMD8 execution there is no known usage model for use of Vector Mask (as there is for PS shaders).

25 — Accesses UAV

Format:	Enable

This field must be set when HS has a UAV access

Programming Notes
This field must not be set when HS Function Enable is disabled.

3DSTATE_HS

24 Include Vertex Handles

Format:	Boolean

If set, all the input Vertex URB handles are included in payloads.
This field is ignored if **HS Function Enable** is DISABLED.

Programming Notes

Programming Restriction:
This field must be set if value if **Vertex URB Entry Read Length** is cleared to zero.

23:19 Dispatch GRF Start Register For URB Data

Format:	U5

Description

Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.
This field is ignored if **HS Function Enable** is DISABLED.

The Dispatch GRF Start Register for URB Data [5] field is used to extend the range of the starting GRF register number to [0,63].

Value	Name	Description
[0,31]		indicating bits [4:0] of the GRF number

Programming Notes

When Include Vertex Handles is set for non-instanced 8_PATCH dispatch of PATCHLIST_30..32 objects, pushed vertex data and/or pushed constants cannot be used as they would need to start in the payload beyond the range of this field (i.e., beyond R31). When Include PrimitiveID is also set, this issue extends to non-instanced 8_PATCH dispatch of PATCHLIST_29..32 objects.

18:17 Dispatch Mode

Format:	U2

This field is unused to set the current thread dispatch mode for the HS stage.

Value	Name	Description
0h	SINGLE_PATCH	HS threads are passed inputs and an output handle associated with a single input patch.
1h	DUAL_PATCH	HS threads are passed inputs and an output handle associated with (up to) two input patches. Patch 0 data is passed in the four lower channels while Patch 1 data (if present) is passed in the four upper channels. Restrictions: Only valid for 4 or fewer input control points (PATCHLIST_4 and below). SW is expected to use MI_TOPOLOGY_FILTER to ensure only patches with the expected # of ICPs are processed by the pipeline when HS is enabled.
2h	8_PATCH	HS threads are passed inputs and an output handle associated with (up to) 8 patches in SIMD8 fashion. See 8_PATCH Thread Payload.

3DSTATE_HS

		3h Reserved
	16:11	**Vertex URB Entry Read Length** Format: U6 Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. This field is ignored if HS Function Enable is DISABLED. Value: [0,63] — Name: **Programming Notes** **Programming Restriction:** This field must be a non-zero value if **Include Vertex Handles** is cleared to zero.
	10	**Reserved** Format: MBZ
	9:4	**Vertex URB Entry Read Offset** Format: U6 Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if HS Function Enable is DISABLED. Value: [0,63] — Name:
	3:1	**Reserved** Format: MBZ
	0	**Include Primitive ID** Format: Enable If set, R1 of the payload is written with Primitive ID value(s). If clear, these Primitive IDs are not included in the payload R1. **Programming Notes** This field is only used when DUAL_PATCH DispatchMode is specified. In SINGLE_PATCH, the single Primitive ID is always passed in R0.
8	31:0	**Reserved** Format: MBZ

3DSTATE_INDEX_BUFFER

3DSTATE_INDEX_BUFFER

Source: RenderCS

Length Bias: 2

This command is used to specify the current IB state used by the VF function. At most one IB is defined and active at any given time.NOTES: The IB must be specified before any RANDOM 3D_PRIMITIVE commands are issued It is possible to have vertex elements source completely from generated ID values and therefore not require any Index Buffer accesses. In this case, VF function will simply ignore the Index Buffer state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_INDEX_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:11	**Reserved** Format: MBZ
	10	**Reserved** Format: MBZ
	9:8	**Index Format** Format: U2 Enumerated type This field specifies the data format of the index buffer. All index values are UNSIGNED. Value: 0h — Name: BYTE

3DSTATE_INDEX_BUFFER

		1h: WORD 2h: DWORD
	7	**Reserved** Format: MBZ
	6:0	**Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this index buffer.
2..3	63:0	**Buffer Starting Address** Format: GraphicsAddress[63:0]Index_Buffer_Entry This field contains the size-aligned (as specified by Index Format) Graphics Address LSBs of the first element of interest within the index buffer. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer. **Programming Notes** Index Buffers can only be allocated in linear (not tiled) graphics memory.
4	31:0	**Buffer Size** Format: U32 Count of bytes This field specifies the size of the buffer in bytes. Index accesses which straddle or go past the end of the buffer will return 0..Note that BufferSize=0 indicates that there is no valid data in the buffer. **Value** / **Name** [0, FFFFFFFFh]

3DSTATE_LINE_STIPPLE

3DSTATE_LINE_STIPPLE

Source: RenderCS

Length Bias: 2

The 3DSTATE_LINE_STIPPLE command is used to specify state variables used in the Line Stipple function.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 08h 3DSTATE_LINE_STIPPLE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n Total Length - 2
1	31	**Modify Enable (Current Repeat Counter, Current Stipple Index)** Format: Enable Modify enable for **Current Repeat Counter** and **Current Stipple Index** fields. **Programming Notes** It is provided only for HW-generated commands as part of context save/restore. SW must initilize the current repeat counter, current stipple count fields if it sets this bit to enable. SW must set this bit to reset the stipple count.
	30	**Reserved** Format: MBZ
	29:21	**Current Repeat Counter** Format: U9 This field sets the HW-internal repeat counter state. SW must initilize it to 1 if the modify enable is set.

3DSTATE_LINE_STIPPLE

	20	**Reserved** Format: MBZ
	19:16	**Current Stipple Index** Format: U4 This field sets the HW-internal stipple pattern index. SW must initialize it to 0 if the modify enable is set.
	15:0	**Line Stipple Pattern** Format: 16 bit mask Bit 15 = most significant bit, Bit 0 = least significant bit Specifies a pattern used to mask out bit specific pixels while rendering lines.
2	31:15	**Line Stipple Inverse Repeat Count** Format: U1.16 Range: [0.00390625, 1.0] Specifies the inverse (truncated) of the repeat count for the line stipple function.
	14:9	**Reserved** Format: MBZ
	8:0	**Line Stipple Repeat Count** Format: U9 Specifies the repeat count for the line stipple function.

Value	Name
[1, 256]	

3DSTATE_MONOFILTER_SIZE

3DSTATE_MONOFILTER_SIZE

Source: RenderCS

Length Bias: 2

This state specifies the size of the filter which is used when filtering in MAPFILTER_MONO mode.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 11h 3DSTATE_MONOFILTER_SIZE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:6	**Reserved** Format: MBZ
	5:3	**Monochrome Filter Width** Format: U3 This field specifies the width of the monochrome filter. It is ignored if the monochrome filter is not enabled. Value: [1,7] — Name:

3DSTATE_MONOFILTER_SIZE		
	2:0	**Monochrome Filter Height** Format: U3 This field specifies the height of the monochrome filter. It is ignored if the monochrome filter is not enabled. **Value** / **Name** [1,7]

3DSTATE_MULTISAMPLE

3DSTATE_MULTISAMPLE

Source: RenderCS

Length Bias: 2

The 3DSTATE_MULTISAMPLE command is used to specify multisample state associated with the current render target/depth buffer.

Programming Notes

It is illegal to render to surfaces with multiple different values of the state fields in this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Dh 3DSTATE_MULTISAMPLE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:6	**Reserved** Format: MBZ
	5	**Pixel Position Offset Enable** Format: Enable Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions. **Programming Notes** Setting this field along with setting the Pixel Location to upper left and number of multisamples to greater than one will cause the device to offset pixel postions by 0.5 both in horizontal and vertical directions.

3DSTATE_MULTISAMPLE

It is to be noted this is done to adjust the pixel co-ordinate system to DX9 like, so any WM_HZ_OP screen space rectangles (eg: legacy HiZ Clear, Resolve etc) generated internally by driver in this mode needs to be aware of this offset adjustment and send the rectangles according to alignment restriction taking this offset adjustment into consideration.
SW can choose to set this bit only for DX9 API. DX10/OGL API's should not have any effect by setting or not setting this bit.

4 Pixel Location

Format:	U1

This field specifies where the device evaluates "pixel" (vs. centroid or sample) values/attributes.

Value	Name	Description
0h	CENTER	Use the pixel center (0.5, 0.5 offset)
1h	UL_CORNER	Use the pixel upper-left corner

Programming Notes

The programming of this field is assumed to be a function of the API being supported. Specifically, it is expected that OpenGL and DX10+ APIs require CENTER selection, while DX9- APIs require UL_CORNER selection.

When 3DSTATE_RASTER::**ForcedSampleCount** is other than NUMRASTSAMPLES_0, this field must be 0h.

3:1 Number of Multisamples

Format:	U3

This field specifies how many samples/pixel exist in all RTs and the Depth Buffer, as log2(#samples). This field is valid regardless of the setting of **Multisample Rasterization Mode**.

Value	Name	Description
0h	1	1 sample/pixel
1h	2	2 samples/pixel
2h	4	4 samples/pixel
3h	8	8 samples/pixel
4h	16	16 samples/pixel
5h-7h	Reserved	

Programming Notes

The setting of this field must match the **Number of Multisamples** field in SURFACE_STATE of all bound render targets.

0 Reserved

Format:	MBZ

3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_OFFSET

Source: RenderCS

Length Bias: 2

The 3DSTATE_POLY_STIPPLE_OFFSET command is used to specify the origin of the repeated screen-space Polygon Stipple Pattern as an X, Y offset from the Color Buffer origin.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 06h 3DSTATE_POLY_STIPPLE_OFFSET Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:13	**Reserved** Format: MBZ
	12:8	**Polygon Stipple X Offset** Format: U5 Specifies a 5 bit x address offset in the poly stipple pattern Value: [0,31] — Name:
	7:5	**Reserved** Format: MBZ

3DSTATE_POLY_STIPPLE_OFFSET		
	4:0	**Polygon Stipple Y Offset** Format: U5 Specifies a 5 bit y address offset in the poly stipple pattern

Value	Name
[0,31]	

3DSTATE_POLY_STIPPLE_PATTERN

3DSTATE_POLY_STIPPLE_PATTERN

Source: RenderCS

Length Bias: 2

The 3DSTATE_POLY_STIPPLE_PATTERN command is used to specify the 32x32 Polygon Stipple Pattern used in the Polygon Stipple function of the WM unit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 07h 3DSTATE_POLY_STIPPLE_PATTERN Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 1Fh Excludes Dword (0,1) Format: =n Total Length - 2
1..32	31:0	**Pattern Row** Format: 32 bit mask Bit 31 = upper left corner, Bit 0 = upper right corner of first row. Specifies a pattern used by Polygon Stipple to mask out specific pixels of every 32x32 area rendered.

3DSTATE_PS_BLEND

3DSTATE_PS_BLEND		
Source:	RenderCS	
Length Bias:	2	

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Dh 3DSTATE_PS_BLEND Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**Alpha To Coverage Enable** Format: Enable If set, indicates that AlphaToCoverage is on RT[0], since this bit must be set the same for all RTs in the MRT case.
	30	**Has Writeable RT** Format: Enable When set indicates the there is at least one non-null RT w/ at least one channel write enabled
	29	**Color Buffer Blend Enable** Format: Enable When set indicates that RT[0] has color buffer blend enabled.

3DSTATE_PS_BLEND

	28:24	**Source Alpha Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Indicates the "source factor" in alpha Color Buffer Blending stage for RT[0]
	23:19	**Destination Alpha Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Indicates the "destination factor" in alpha Color Buffer Blending stage for RT[0]
	18:14	**Source Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Indicates the "source factor" in Color Buffer Blending stage for RT[0]
	13:9	**Destination Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Indicates the "destination factor" in Color Buffer Blending stage for RT[0]
	8	**Alpha Test Enable** Format: Enable Indicates the AlphaTestEnable for RT[0]
	7	**Independent Alpha Blend Enable** Format: Enable Indicates the Independent Alpha Blend Enable for RT[0] When enabled, the other fields in this instruction control the combination of the alpha components in the Color Buffer Blend stage. When disabled, the alpha components are combined in the same fashion as the color components.
	6:0	**Reserved** Format: MBZ

3DSTATE_PS

3DSTATE_PS

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 20h 3DSTATE_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0Ah Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:6	**Kernel Start Pointer 0** Format: InstructionBaseOffset[63:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel[0]. This pointer is relative to the **Instruction Base Address.**
	5:0	**Reserved** Format: MBZ
3	31	**Single Program Flow** Single Program Flow (SPF) specifies the initial condition of the kernel program as either a single program flow (SIMDnxm with m = 1) or as multiple program flows (SIMDnxm with m > 1). See CR0 description in ISA Execution Environment. Value / Name / Description 0h / Multiple / Multiple Program Flows 1h / Single / Single Program Flows

3DSTATE_PS

30 Vector Mask Enable

Format:	Enable

When SPF=0, Vector Mask Enable (VME) specifies which mask to use to initialize the initial channel enables. When SPF=1, VME specifies which mask to use to generate execution channel enables.

Value	Name	Description
0h	Dmask	Channels are enabled based on the dispatch mask
1h	Vmask	Channels are enabled based on the vector mask

29:27 Sampler Count

Specifies how many samplers (in multiples of 4) the vertex shader 0 kernel uses. Used only for prefetching the associated sampler state entries.

Value	Name	Description
[0,4]		
0h	No Samplers	no samplers used
1h	1-4 Samplers	between 1 and 4 samplers used
2h	5-8 Samplers	between 5 and 8 samplers used
3h	9-12 Samplers	between 9 and 12 samplers used
4h	13-16 Samplers	between 13 and 16 samplers used
5h-7h		Reserved

26 Single Precision Denormal Mode

Specifies the single precision denornal mode used by the dispatched thread.

Value	Name	Description
0h	Flushed to Zero	Single Precision denormals are flushed to zero
1h	Retained	Single Precision denormals are retained

25:18 Binding Table Entry Count

Description
Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. **Note:** For kernels using a large number of binding table entries, it may be advantageous to set this field to zero to avoid prefetching too many entries and thrashing the state cache. This field is ignored if [PS Function Enable] is DISABLED.
When [HW Generated Binding Table] bit is enabled: This field indicates which cache lines (512bit units - 32 Binding Table Entry section) should be fetched. Each bit in this field corresponds to a cache line. Only the 1st 4 non-zero Binding Table entries of each 32 Binding Table entry section prefetched will have its surface state prefetched. See 3D Pipeline for more information.

Programming Notes
When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be

3DSTATE_PS

generated at JIT time.

17 Thread Dispatch Priority

Specifies the priority of the thread for dispatch.

Value	Name	Description
0h	Normal	Normal Priority
1h	High	High Priority

16 Floating Point Mode

Specifies the floating point mode used by the dispatched thread.

Value	Name	Description
0h	IEEE-754	Use IEEE-754 rules
1h	Alternate	Use alternate rules

15:14 Rounding Mode

Specifies the rounding mode used by the dispatched thread.

Value	Name	Description
0h	RTNE	Round to Nearest Even
1h	RU	Round toward +infinity
2h	RD	Round toward -infinity
3h	RTZ	Round toward zero

13 Illegal Opcode Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.

12 Reserved

Format:	MBZ

11 Mask Stack Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.

10:8 Reserved

Format:	MBZ

7 Software Exception Enable

Format:	Enable

This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.

6:0 Reserved

3DSTATE_PS

		Format: MBZ
4..5	63:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[63:10]ScratchSpace Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the General State Base Address. **Programming Notes** Scratch Space per slice is computed based on 4 sub-slices. SW must allocate scratch space enough so that each slice has 4 slices allowed.
	9:4	**Reserved** Format: MBZ
	3:0	**Per Thread Scratch Space** Format: U4 Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the Maximum Number of Threads each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space. **Value** | **Name** [0,11] | indicating [1k bytes, 2M bytes] in powers of two
6	31:23	**Maximum Number of Threads Per PSD** Format: U8-1 Representing Thread Count Range = [1, 63] --> [2, 64] threads. Specifies the maximum number of simultaneous threads allowed to be active per PSD. With one PSD per subslice, the maximum number of PS threads in the system is this field multiplied by the number of enabled subslices in the current configuration. **Programming Notes** If this field is changed between 3DPRIMITIVE commands, a PIPE_CONTROL command with **Stall at Pixel Scoreboard** set is required to be issued.
	22:12	**Reserved** Format: MBZ
	11	**Push Constant Enable** Format: Enable This field must be enabled if the sum of the **PS Constant Buffer [3:0] Read Length** fields in 3DSTATE_CONSTANT_PS is nonzero, and must be disabled if the sum is zero.
	10	**Reserved** Format: MBZ
	9	**Reserved** Format: MBZ

3DSTATE_PS

8 — Render Target Fast Clear Enable

Format:	Enable

This field is set to enable fast clear of the bound render targets. See "Render Target Fast Clear" for restrictions on enabling this field.

Programming Notes
When this bit is set, corresponding BTI for the render target that is being cleared must be equal to 0.
When this bit is set, RENDER_SURFACE_STATE type must not be NULL.

7:6 — Render Target Resolve Type

Format:	U2 Enumerated Type

Specifies what type of Render Target Resolve is needed for the surface to be consumed properly by the end Client. Programming notes below.

Value	Name	Description	Programming Notes
0h	RESOLVE_DISABLED	No Resolve Needed	
1h	RESOLVE_PARTIAL	Partial resolve is for resolving RT for clear values i.e. it leaves no cache lines at implied clear value.	Display engine does not support unresolved clear values in the display buffer, hence this resolve is required before binding any compressed RT to the display via flip commands. If non 0/1 clear values are used, software must perform partial resolve before binding cleared render target as texture.
2h	Reserved		
3h	RESOLVE_FULL	Full Resolve is for Resolving RT for Clear/Compressed to Uncompressed State	

Programming Notes
When this bit is set, corresponding BTI for the render target that is being resolved must be equal to 0.
When this bit is set, RENDER_SURFACE_STATE type must not be NULL.

5 — Reserved

Format:	MBZ

4:3 — Position XY Offset Select

Format:	U2 Enumerated Type

This field specifies if/what Position XY Offset values are passed in the PS payload. Note that these are per-slot (pixel|sample) offsets, and therefore separate from the subspan XY coordinates passed in R1.

3DSTATE_PS

Value	Name	Description
0h	POSOFFSET_NONE	No Position XY Offsets are included in the PS payload.
1h	Reserved	
2h	POSOFFSET_CENTROID	Position XY Offsets will be passed in the PS payload, and these will reflect the Centroid position(s).
3h	POSOFFSET_SAMPLE	Position XY Offsets will be passed in the PS payload, and these will reflect the multisample position(s).

Programming Notes
SW Recommendation: If the PS kernel needs the Position Offsets to compute a Position XY value, this field should match Position ZW Interpolation Mode to ensure a consistent position.xyzw computation
If the PS kernel does not need the Position XY Offsets to compute a Position Value, then this field should be programmed to POSOFFSET_NONE, as the PS kernel should be using the various barycentric inputs to evaluate other-than-position attributes. However, this field can be used to pass Centroid or Sample offsets in the payload for special test modes (e.g., where barycentric coordinates are computed in the PS vs. being HW-generated and passed in the payload).
MSDISPMODE_PERSAMPLE is required in order to select POSOFFSET_SAMPLE.

2

32 Pixel Dispatch Enable

Format:	Enable

Enables the Windower to dispatch 8 subspans in one payload. Variable Pixel Dispatch in Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.

Programming Notes
When NUM_MULTISAMPLES = 16 or FORCE_SAMPLE_COUNT = 16, SIMD32 Dispatch must not be enabled for PER_PIXEL dispatch mode.

1

16 Pixel Dispatch Enable

Format:	Enable

Enables the Windower to dispatch 4 subspans in one payload. Variable Pixel Dispatch in Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.

0

8 Pixel Dispatch Enable

Format:	Enable

Enables the Windower to dispatch 2 subspans from 1 object (polygon) in one payload. Variable Pixel Dispatch in Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.

Programming Notes
When Render Target Fast Clear Enable is ENABLED or Render Target Resolve Type = RESOLVE_PARTIAL or RESOLVE_FULL, this bit must be DISABLED.

3DSTATE_PS

7	31:23	**Reserved** Format: MBZ
	22:16	**Dispatch GRF Start Register For Constant/Setup Data 0** Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[0]. Value: [0,127] — Name:
	15	**Reserved** Format: MBZ
	14:8	**Dispatch GRF Start Register For Constant/Setup Data 1** Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[1]. Value: [0,127] — Name:
	7	**Reserved** Format: MBZ
	6:0	**Dispatch GRF Start Register For Constant/Setup Data 2** Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[2]. Value: [0,127] — Name:
8..9	63:6	**Kernel Start Pointer 1** Format: InstructionBaseOffset[63:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in kernel[1]. This pointer is relative to the Instruction Base Address.
	5:0	**Reserved** Format: MBZ
10..11	63:6	**Kernel Start Pointer 2** Format: InstructionBaseOffset[63:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in kernel[2]. This pointer is relative to the **Instruction Base Address**.
	5:0	**Reserved** Format: MBZ

3DSTATE_PS_EXTRA

3DSTATE_PS_EXTRA

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4fh 3DSTATE_PS_EXTRA Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**Pixel Shader Valid** Format: Enable When set indicates a valid pixel shaderWhen this bit clear the rest of this command should also be clear.
	30	**Pixel Shader Does not write to RT** Format: Enable When set indicates the pixel shader does not write to render target. **Programming Notes** When Pixel Shader writes to UAV but does not write to RT, a dummy render target write is required to convey EOT to the PS dispatch function. Hence, this bit must be reset in this case. Whenever, there is a render target write message even to the NULL render target, this bit must be reset. When Pixel Shader Kills Pixel is set, SW must perform a dummy render target write from the shader and not set this bit, so that Occlusion Query is correct.

3DSTATE_PS_EXTRA

29 **oMask Present to Render Target**

Format:	Enable

This bit is inserted in the PS payload header and made available to the DataPort (either via the message header or via header bypass) to indicate that oMask data from the shader (one or two phases) is included in Render Target Write messages. If present, the oMask data is used to mask off samples.

28 **Pixel Shader Kills Pixel**

Format:	Enable

This bit, if ENABLED, indicates that the PS kernel has the ability to kill (discard) pixels or samples, other than due to depth or stencil testing. This bit is required to be ENABLED in the following situations:

- The API pixel shader program contains "killpix" or "discard" instructions, or other code in the pixel shader kernel that can cause the final pixel mask to differ from the pixel mask received on dispatch.

27:26 **Pixel Shader Computed Depth Mode**

Format:	U2 Enumerated Type

This field specifies the computed depth mode for the pixel shader.

Value	Name	Description
0h	PSCDEPTH_OFF	Pixel shader does not compute depth
1h	PSCDEPTH_ON	Pixel shader computes depth with no guarantee as to its value
2h	PSCDEPTH_ON_GE	
3h	PSCDEPTH_ON_LE	Pixel shader computes depth and guarantees that oDepth <= SourceDepth If the Position ZW interpolation mode in 3DSTATE_WM does not match the DX Multisample Rasterization mode in 3DSTATE_RASTER, HW will internally convert to PSCDEPTH_ON.

Programming Notes

If this field is set to any value other than PSCDEPTH_OFF, a multi-phase shader (i.e. dispatch RATE_COARSE or RATE_PIXEL with pixel/sample loops or sample loop respectively) must output depth and render targets only at the last phase.

25 **Force Computed Depth**

Format:	Enable

Programming Notes

This field should be left DISABLED. This field should not be tested for functional validation.

24 **Pixel Shader Uses Source Depth**

Format:	Enable

This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be

3DSTATE_PS_EXTRA

		passed in the payload. The source depth value is interpolated according to the Position ZW Interpolation Mode state.
	23	**Pixel Shader Uses Source W** Format: Enable This bit, if ENABLED, indicates that the PS kernel requires the interpolated source W value (vPos.w) to be passed in the payload. The W value is interpolated according to the Position ZW Interpolation Mode state.
	22	**Reserved** Format: MBZ
	21:18	**Reserved** Format: MBZ
	17	**Reserved** Format: MBZ
	16:11	**Reserved** Format: MBZ
	10	**Reserved** Format: MBZ
	9	**Reserved**
	8	**Attribute Enable** Format: Enable This field must be enabled if the Number of SF Output Attributes field in 3DSTATE_SBE is nonzero, and must be disabled if that field is zero.
	7	**Pixel Shader Disables Alpha To Coverage** Format: Enable When set indicates the pixel shader AlphaToCoverage should be disabled due to oMask output. The setting of this bit is API dependent.
	6	**Pixel Shader Is Per Sample** Format: Enable This bit, when ENABLED, indicates that the pixel shader is dispatched at the per sample shading rate. If the bit is DISABLED, the pixel shader is dispatched at the per pixel rate. **Programming Notes** This bit must NOT be set when PS is used to do clear MSRTs with Fast Clear Optimization Enabled. When NUMRASTSAMPLES is other than 1 (i.e. when FORCED_SAMPLE_COUNT is greater than 1, hence Target Independent Rasterization is enabled) , this bit MUST be DISABLED.
	5	**Pixel Shader Computes Stencil** Format: Enable This field when set indicates that the pixel shader computes the stencil reference value.

3DSTATE_PS_EXTRA

Programming Notes

If this field is ENABLED, a multi-phase shader (i.e. dispatch RATE_COARSE or RATE_PIXEL with pixel/sample loops or sample loop respectively) must output stencil and render targets only at the last phase.

4 **Reserved**

Format:	MBZ

3 **Pixel Shader Pulls Bary**

Format:	Enable

This bit indicates if Pixel Shader uses Pull Bary i.e. PI message. If this bit is reset, PS does not do Pull Bary.

2 **Pixel Shader Has UAV**

Format:	Enable
Format:	U1 Enumerated Type

This field when set indicates that the pixel shader has a UAV attached to it.

1:0 **Input Coverage Mask State**

Format:	U2

This field indicates the type of input coverage mask that the PS kernel requires to be passed in the payload.

Value	Name	Description
0h	NONE	Pixel shader does not use input coverage masks.
1h	NORMAL	Input Coverage masks based on outer conservatism and factors in SAMPLE_MASKs. If Pixel is conservatively covered, all samples are enabled.
2h	INNER_CONSERVATIVE	Input Coverage masks based on inner conservatism and factors in SAMPLE_MASKs. If Pixel is conservatively fully covered all samples are enabled.
3h	DEPTH_COVERAGE	Input coverage masks are computed after factoring depth/stencil test results, only if Early Depth Stencil Test is enabled. If Early Depth Stencil Test is not enabled, HW uses NORMAL Input Coverage Masks.

3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS

Source: RenderCS

Length Bias: 2

This command sets up the URB configuration for DS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_DS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The 3DSTATE_CONSTANT_DS must be committed as state prior to shaders generating thread payloads after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_DS.

When gather at set shader is disabled, 3DSTATE_CONSTANT command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the 3DSTATE_CONSTANT command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER_DS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_DS if 3DSTATE_CONSTANT_VS command is committed on a binding table pointer due to Gather at Set Shader feature.

DWord	Bit	Description
0 **Programming Notes:** Workaround: This command must be followed by a PIPE_CONTROL with CS Stall bit set.,	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_PUSH_CONSTANT_ALLOC_DS Format: OpCode
	15:8	**Reserved** Format: MBZ

3DSTATE_PUSH_CONSTANT_ALLOC_DS

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the DS constant buffer into the URB. Value: [0,31] — Name: (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the DS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for DS. Value: [0,32] — Name: (0KB - 32KB) Increments of 2KB

3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_PUSH_CONSTANT_ALLOC_GS

Source: RenderCS

Length Bias: 2

This command sets up the URB configuration for GS Push Constant Buffer.

Programming Notes

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_GS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines.
- The 3DSTATE_CONSTANT_GS must be committed as state prior to shaders generating thread payloads after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_GS.

See Push Constant URB Allocation section for more details.

When gather at set shader is disabled, 3DSTATE_CONSTANT command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the 3DSTATE_CONSTANT command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER_GS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_GS if 3DSTATE_CONSTANT_VS command is committed on a binding table pointer due to Gather at Set Shader feature.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 15h 3DSTATE_PUSH_CONSTANT_ALLOC_GS Format: OpCode
	15:8	**Reserved** Format: MBZ

3DSTATE_PUSH_CONSTANT_ALLOC_GS

	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 0h / 3DSTATE_PUSH_CONSTANT_ALLOC_GS **[Default]** / Excludes DWord (0,1)
1	31:21	**Reserved** Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the GS constant buffer into the URB. **Value** / **Name** [0,31] / (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the GS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for GS. **Value** / **Name** [0,32] / (0KB - 32KB) Increments of 2KB

3DSTATE_PUSH_CONSTANT_ALLOC_HS

3DSTATE_PUSH_CONSTANT_ALLOC_HS

Source: RenderCS

Length Bias: 2

This command sets up the URB configuration for HS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_HS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The 3DSTATE_CONSTANT_HS must be committed as state prior to shaders generating thread payloads after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_HS.

When gather at set shader is disabled, 3DSTATE_CONSTANT command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the 3DSTATE_CONSTANT command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER_HS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_VS if 3DSTATE_CONSTANT_HS command is committed on a binding table pointer due to Gather at Set Shader feature.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 13h 3DSTATE_PUSH_CONSTANT_ALLOC_HS Format: OpCode
	15:8	**Reserved** Format: MBZ

3DSTATE_PUSH_CONSTANT_ALLOC_HS

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the HS constant buffer into the URB. Value: [0,31] — Name: (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the HS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for HS. Value: [0,32] — Name: (0KB - 32KB) Increments of 2KB

3DSTATE_PUSH_CONSTANT_ALLOC_PS

3DSTATE_PUSH_CONSTANT_ALLOC_PS

Source: RenderCS

Length Bias: 2

This command sets up the URB configuration for PS Push Constant Buffer.

Programming Notes

Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_PS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation** section for more details.
- The 3DSTATE_CONSTANT_PS must be committed as state prior to shaders generating thread payloads after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_PS.

When gather at set shader is disabled, 3DSTATE_CONSTANT command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the 3DSTATE_CONSTANT command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER_PS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_PS if 3DSTATE_CONSTANT_VS command is committed on a binding table pointer due to Gather at Set Shader feature.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 16h 3DSTATE_PUSH_CONSTANT_ALLOC_PS Format: OpCode
	15:8	**Reserved** Format: MBZ

3DSTATE_PUSH_CONSTANT_ALLOC_PS

	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the PS constant buffer into the URB. Value: [0,31] — Name: (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the PS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for PS. Value: [0,32] — Name: (0KB - 32KB) Increments of 2KB

3DSTATE_PUSH_CONSTANT_ALLOC_VS

3DSTATE_PUSH_CONSTANT_ALLOC_VS

Source: RenderCS

Length Bias: 2

This command sets up the URB configuration for VS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_VS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The 3DSTATE_CONSTANT_VS must be committed as state prior to shaders generating thread payloads after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_VS.

When gather at set shader is disabled, 3DSTATE_CONSTANT command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the 3DSTATE_CONSTANT command is a 3DSTATE_BINDING_TABLE_POINTER command.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	**Command SubType**	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	**3D Command Opcode**	
		Default Value:	1h 3DSTATE_NONPIPELINED
		Format:	OpCode
	23:16	**3D Command Sub Opcode**	
		Default Value:	12h 3DSTATE_PUSH_CONSTANT_ALLOC_VS
		Format:	OpCode
	15:8	**Reserved**	
		Format:	MBZ
	7:0	**DWord Length**	
		Default Value:	0h Excludes DWord (0,1)
		Format:	=n Total Length - 2

3DSTATE_PUSH_CONSTANT_ALLOC_VS

1	31:21	**Reserved** Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the VS constant buffer into the URB. Value: [0,31] — Name: (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the VS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for VS. Value: [0,32] — Name: (0KB - 32KB) Increments of 2KB

3DSTATE_RASTER

3DSTATE_RASTER

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 50h 3DSTATE_RASTER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 03h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:28	**Reserved** Format: MBZ
	27	**Reserved** Format: MBZ
	26	**Viewport Z Far Clip Test Enable** Format: Enable This field is used to control whether the Viewport Z Far extent is considered in VertexClipTest.
	25	**Reserved** Format: MBZ
	24	**Conservative Rasterization Enable** Format: Enable This field when set enables conservative rasterization rules for all primitives except rectangles,

3DSTATE_RASTER

points and lines. For rectangle, points and lines, setting this bit is ignored by hardware.

Programming Notes
This bit must not be set for primitives with poly-stippling enabled. When this bit is set, sampling mode must be set to "Centre" sampling i.e 3DSTATE_MULTISAMPLE::Pixel Location set to CENTER

23:22 **API Mode**

Software sets this field according to the API's version. These bits are set for DX9 or OGL/DX10.0/DX10.1+/DX11.1 per the following values.

Value	Name
0h	DX9/OGL
1h	DX10.0
2h	DX10.1+
3h	Reserved

21 **Front Winding**

Determines whether a triangle object is considered "front facing" if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.

Value	Name	Description
0h	Clockwise	FRONTWINDING_CW
1h	Counter Clockwise **[Default]**	FRONTWINDING_CCW

20:18 **Forced Sample Count**

Format:	U3 Enumerated Type

This field specifies how many samples/pixel exist for RT Independent Rasterization

Value	Name	Description
0h	NUMRASTSAMPLES_0	No RT Independent Rasterization
1h	NUMRASTSAMPLES_1	1 rast-sample/pixel
2h	NUMRASTSAMPLES_2	2 rast-samples/pixel
3h	NUMRASTSAMPLES_4	4 rast-samples/pixel
4h	NUMRASTSAMPLES_8	8 rast-samples/pixel
5h	NUMRASTSAMPLES_16	16 rast-samples/pixel
6h-7h	Reserved	

Programming Notes
When 3DSTATE_MULTISAMPLE::Number of Multisamples !=NUMSAMPLES_1, this field mus be either NUMRASTSAMPLES_0 or NUMRASTSAMPLES_1. When 3DSTATE_MULTISAMPLE::Number of Multisamples == NUMSAMPLES_1, this field mus not be NUMRASTSAMPLES_1.

3DSTATE_RASTER

17:16 Cull Mode

Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points or rectangles.

Value	Name	Description
0h	CULLMODE_BOTH	All triangles are discarded (i.e., no triangle objects are drawn)
1h	CULLMODE_NONE **[Default]**	No triangles are discarded due to orientation
2h	CULLMODE_FRONT	Triangles with a front-facing orientation are discarded
3h	CULLMODE_BACK	Triangles with a back-facing orientation are discarded

Programming Notes
Orientation determination is based on the setting of the Front Winding state.

15 Reserved

Format:	MBZ

14 Force Multisampling

This field provides a work around override for the computation of SF_INT::Multisample Rasterization Mode and WM_INT::Multisample Rasterization Mode.

Value	Name	Description
0h	Normal	Multisampling mode is computed by HW according to formula for signal SF_INT::Multisample Rasterization Mode and WM_INT::Multisample Rasterization Mode in vol2a.11 3D Pipeline Windower > Windower Pipelined State > 3DSTATE_WM > 3DSTATE_WM.
1h	Force	Forces the DX Multisampling mode to be used directly

13 Smooth Point Enable

Format:	Enable

Software sets this according to API. When OGL and smooth point rasterization is required, this bit must be set. HW ignores this bit for primitives other than points.

12 DX Multisample Rasterization Enable

Format:	Enable

Software sets this according to the API's multisample enable

Programming Notes
This state only effects how the SF_INT/WM_INT::Multisample Rasterization Mode are set depending on some other states. This state mainly modifies the how the line rendering is done by setting SF_INT/WM_INT::Multisample Rasterization Mode to either OFF* or ON* . Please refer to table under SF_INT::Multisample Rasterization Mode.

11:10 DX Multisample Rasterization Mode

Format:	U2 enumerated type

This field determines whether multisample rasterization is turned on/off, and how the pixel

3DSTATE_RASTER

sample point(s) are defined. Software sets this according to the API's multisample state setting (if any)

Value	Name
0h	MSRASTMODE_ OFF_PIXEL
1h	MSRASTMODE_ OFF_PATTERN
2h	MSRASTMODE_ ON_PIXEL
3h	MSRASTMODE_ ON_PATTERN

Programming Notes
This field is used to directly set the SF_INT/WM_INT::Multisample Rasterization Mode when DX Multisample Rasterization Enable is set. Please refer to equation of SF_INT::Multisample Rasterization Mode.

9 **Global Depth Offset Enable Solid**

Format:	Enable

Enables computation and application of Global Depth Offset for SOLID objects.

8 **Global Depth Offset Enable Wireframe**

Format:	Enable

Enables computation and application of Global Depth Offset when triangles are rendered in WIREFRAME mode.

7 **Global Depth Offset Enable Point**

Format:	Enable

Enables computation and application of Global Depth Offset when triangles are rendered in POINT mode.

6:5 **Front Face Fill Mode**

Format:	U2 enumerated type

This state controls how front-facing triangle and rectangle objects are rendered.

Value	Name	Description
0h	SOLID	Any triangle or rectangle object found to be front-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.
1h	WIREFRAME	Any triangle object found to be front-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).
2h	POINT	Any triangle object found to be front-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags).
3h	Reserved	

3DSTATE_RASTER

4:3 Back Face Fill Mode

Format:	U2 enumerated type

This state controls how back-facing triangle and rectangle objects are rendered.

Value	Name	Description
0h	SOLID	Any triangle or rectangle object found to be back-facing is rendered as a solid object. This setting is required when rendering rectangle (RECTLIST) objects.
1h	WIREFRAME	Any triangle object found to be back-facing is rendered as a series of lines along the triangle boundaries (as determined by the topology type and controlled by the vertex EdgeFlags).
2h	POINT	Any triangle object found to be back-facing is rendered as a set of point primitives at the triangle vertices (as determined by the topology type and controlled by the vertex EdgeFlags).
3h	Reserved	

2 Antialiasing Enable

Format:	Enable

This field enables "alpha-based" line antialiasing.

Programming Notes
This field must be disabled if any of the render targets have integer (UINT or SINT) surface format.

1 Scissor Rectangle Enable

Format:	Enable

Enables operation of Scissor Rectangle.

0 Viewport Z Near Clip Test Enable

Format:	Enable

This field is used to control whether the Viewport Z Near extent is considered in VertexClipTest.

2 31:0 Global Depth Offset Constant

Format:	IEEE_Float

Specifies the constant term in the Global Depth Offset function.

3 31:0 Global Depth Offset Scale

Format:	IEEE_Float

Specifies the scale term used in the Global Depth Offset function.

4 31:0 Global Depth Offset Clamp

Format:	IEEE_Float

Specifies the clamp term used in the Global Depth Offset function.

3DSTATE_RS_CONSTANT_POINTER

3DSTATE_RS_CONSTANT_POINTER

Source: RenderCS

Length Bias: 2

The 3DSTATE_RS_CONSTANT_POINTER command allows the driver to program an indirect address to either store or load the global section of the RS constant buffers.

Programming Notes

This command will be treated as a NOOP if the Dx9 Constant Buffer Pool is not enabled by the 3DSTATE_DX9_CONST_POOL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 54h 3DSTATE_RS_CONSTANT_POINTER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**Reserved** Format: MBZ
	30:28	**Shader Select** Format: U3 Specifies which shaders global register to load or store.

Value	Name	Description
0h	VS	Select VS Global Constants
1h-3h	Reserved	
4h	PS	Select PS Global Constants

3DSTATE_RS_CONSTANT_POINTER

		5h-7h: Reserved
	27:13	**Reserved** Format: MBZ
	12	**Operation Load or Store** When this bit is set, the resource streamer will load the global constants to the pointer to the global buffer in the hardware based on the **Shader Select** bit. When this bit is clear, the resource streamer will store the global constants to the pointer from the global buffer in the hardware based on the **Shader Select** bit. **Value** / **Name** / **Description** 0h / Store / Store buffer to memory 1h / Load / Load buffer from memory
	11:0	**Reserved** Format: MBZ
2	31:6	**Global Constant Buffer Address** Format: GraphicsAddress[31:6] Specifies the Cache Line Aligned Graphics address of location in memory of where to load or store a copy of the global constant buffer.
	5:0	**Reserved** Format: MBZ
3	31:0	**Global Constant Buffer Address High** Format: GraphicsAddress[63:32] Specifies the Upper portion of the graphics address location in memory of where to load or store a copy of the global constant buffer. Upper 16 bits MBZ.

3DSTATE_SAMPLE_MASK

3DSTATE_SAMPLE_MASK

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 18h 3DSTATE_SAMPLE_MASK Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:16	**Reserved** Format: MBZ
	15:0	**Sample Mask** Format: 16 bit mask Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_MULTISAMPLE) A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection. This mask must be ignored for centroid selection when RTIR is enabled i.e. Forced_Sample_Count > 0. **Programming Notes** • If **Number of Multisamples** is NUMSAMPLES_1, bits 15:1 of this field will be zeroed by HW. • If N**Number of Multisamples** is NUMSAMPLES_2, bits 15:2 of this field will be zeroed by HW. • If **Number of Multisamples** is NUMSAMPLES_4, bits 15:4 of this field will be zeroed by

3DSTATE_SAMPLE_MASK		
		HW. • If **Number of Multisamples** is NUMSAMPLES_8, bits 15:8 of this field will be zeroed by HW. When pixel shader writes to UAV but does not have actual render target write (i.e. no RT is bound to pixel shader, eventhough, RT write message is sent for EOT), appropriate SAMPLE_MASK must be all set depending on Number of Multisamples.

3DSTATE_SAMPLE_PATTERN

3DSTATE_SAMPLE_PATTERN

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLE_PATTERN command is used to specify the sample offsets for all multisample sample modes.

The set of offset used will be selected based on the multisample mode. This is non-pipelined state.

Programming Notes

When programming the sample offsets (for NUMSAMPLES_4 or _8 and MSRASTMODE_xxx_PATTERN), the order of the samples 0 to 3 (or 7 for 8X, or 15 for 16X) must have monotonically increasing distance from the pixel center. This is required to get the correct centroid computation in the device.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_SAMPLE_PATTERN Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 7 Excludes Dword (0,1) Format: =n Total Length - 2
1	31:28	**16x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	27:24	**16x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	19:16	**16x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_4 , _8 or _16. Range: [0,0.9375]
	15:12	**16x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_2, _4 , _8 or _16. Range: [0,0.9375]
	11:8	**16x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	7:4	**16x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	3:0	**16x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

2	31:28	**16x Sample7 X Offset** Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample7 Y Offset** Format: U0.4 Subpixel Y offset of Sample 7 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample6 X Offset** Format: U0.4 Subpixel X offset of Sample 6 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]
	19:16	**16x Sample6 Y Offset** Format: U0.4 Subpixel Y offset of Sample 6 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	15:12	**16x Sample5 X Offset** Format: U0.4 Subpixel X offset of Sample 5 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	11:8	**16x Sample5 Y Offset** Format: U0.4 Subpixel Y offset of Sample 5 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]
	7:4	**16x Sample4 X Offset** Format: U0.4 Subpixel X offset of Sample 4 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	3:0	**16x Sample4 Y Offset** Format: U0.4 Subpixel Y offset of Sample 4 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
3	31:28	**16x Sample11 X Offset** Format: U0.4 Subpixel X offset of Sample 11 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample11 Y Offset** Format: U0.4 Subpixel Y offset of Sample 11 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16. Range: [0,0.9375]
	23:20	**16x Sample10 X Offset** Format: U0.4 Subpixel X offset of Sample 10 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	19:16	**16x Sample10 Y Offset** Format: U0.4 Subpixel Y offset of Sample 10 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16 Range: [0,0.9375]
	15:12	**16x Sample9 X Offset** Format: U0.4 Subpixel X offset of Sample 9 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	11:8	**16x Sample9 Y Offset** Format: U0.4 Subpixel Y offset of Sample 9 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN		
	7:4	**16x Sample8 X Offset** Format: U0.4 Subpixel X offset of Sample 8 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	3:0	**16x Sample8 Y Offset** Format: U0.4 Subpixel Y offset of Sample 8 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
4	31:28	**16x Sample15 X Offset** Format: U0.4 Subpixel X offset of Sample 15 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample15 Y Offset** Format: U0.4 Subpixel Y offset of Sample 15 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample14 X Offset** Format: U0.4 Subpixel X offset of Sample 14 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16. Range: [0,0.9375]
	19:16	**16x Sample14 Y Offset** Format: U0.4 Subpixel Y offset of Sample 14 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16 Range: [0,0.9375]
	15:12	**16x Sample13 X Offset** Format: U0.4 Subpixel X offset of Sample 13 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	11:8	**16x Sample13 Y Offset** Format: U0.4 Subpixel Y offset of Sample 13 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	7:4	**16x Sample12 X Offset** Format: U0.4 Subpixel X offset of Sample 12 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_16. Range: [0,0.9375]
	3:0	**16x Sample12 Y Offset** Format: U0.4 Subpixel Y offset of Sample 12 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
5	31:28	**8x Sample7 X Offset** Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample7 Y Offset** Format: U0.4 Subpixel Y offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample6 X Offset** Format: U0.4 Subpixel X offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	19:16	**8x Sample6 Y Offset** Format: U0.4 Subpixel Y offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	15:12	**8x Sample5 X Offset** Format: U0.4 Subpixel X offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample5 Y Offset** Format: U0.4 Subpixel Y offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample4 X Offset** Format: U0.4 Subpixel X offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample4 Y Offset** Format: U0.4 Subpixel Y offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
6	31:28	**8x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	19:16	**8x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	15:12	**8x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
7	31:28	**4x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	27:24	**4x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	23:20	**4x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	19:16	**4x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	15:12	**4x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	11:8	**4x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	7:4	**4x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	3:0	**4x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
8	31:24	**Reserved** Format: MBZ
	23:20	**1x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 1x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	19:16	**1x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 1x mode. Range: [0,0.9375]
	15:12	**2x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	11:8	**2x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	7:4	**2x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	3:0	**2x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]

3DSTATE_SAMPLER_PALETTE_LOAD0

3DSTATE_SAMPLER_PALETTE_LOAD0

Source: RenderCS

Length Bias: 2

Description
The 3DSTATE_SAMPLER_PALETTE_LOAD0 instruction is used to load 32-bit values into the first texture palette. The texture palette is used whenever a texture with a paletted format (containing "Px [palette0]") is referenced by the sampler.
This instruction is used to load all or a subset of the 256 entries of the first palette. Partial loads always start from the first (index 0) entry.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 02h 3DSTATE_SAMPLER_PALETTE_LOAD0 Format: Opcode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length = 1 + entryCount - 2 Value / Name / Description: [0,255] / Range / 1-256 Entries
1..n	31:0	**Entry** Format: PALETTE_ENTRY

3DSTATE_SAMPLER_PALETTE_LOAD1

3DSTATE_SAMPLER_PALETTE_LOAD1

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_PALETTE_LOAD1 instruction is used to load 32-bit values into the second texture palette. The second texture palette is used whenever a texture with a paletted format (containing "Px...[palette1]") is referenced by the sampler.This instruction is used to load all or a subset of the 256 entries of the second palette. Partial loads always start from the first (index 0) entry.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ch 3DSTATE_SAMPLER_PALETTE_LOAD1 Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1..n	31:24	**Palette Alpha[0:N-1]** Format: U8 Alpha channel loaded into the Nth entry of the texture color palette.
	23:16	**Palette Red[0:N-1]** Format: U8 Alpha channel loaded into the Nth entry of the texture color palette.

3DSTATE_SAMPLER_PALETTE_LOAD1		
	15:8	**Palette Green[0:N-1]** Format: U8 Alpha channel loaded into the Nth entry of the texture color palette.
	7:0	**Palette Blue[0:N-1]** Format: U8 Alpha channel loaded into the Nth entry of the texture color palette.

3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_STATE_POINTERS_DS command is used to define the location of DS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Dh 3DSTATE_SAMPLER_STATE_POINTERS_DS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Pointer to DS Sampler State** Format: DynamicStateOffset[31:5]SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the DS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_STATE_POINTERS_GS command is used to define the location of GS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Eh 3DSTATE_SAMPLER_STATE_POINTERS_GS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Pointer to GS Sampler State** Format: DynamicStateOffset[31:5]SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the GS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_SAMPLER_STATE_POINTERS_HS

3DSTATE_SAMPLER_STATE_POINTERS_HS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_STATE_POINTERS_HS command is used to define the location of HS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Ch 3DSTATE_SAMPLER_STATE_POINTERS_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Pointer to HS Sampler State** Format: DynamicStateOffset[31:5]SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the HS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_SAMPLER_STATE_POINTERS_PS

3DSTATE_SAMPLER_STATE_POINTERS_PS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_STATE_POINTERS_PS command is used to define the location of PS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Fh 3DSTATE_SAMPLER_STATE_POINTERS_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Pointer to PS Sampler State** Format: DynamicStateOffset[31:5]SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the PS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_SAMPLER_STATE_POINTERS_VS

3DSTATE_SAMPLER_STATE_POINTERS_VS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLER_STATE_POINTERS_VS command is used to define the location of VS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Bh 3DSTATE_SAMPLER_STATE_POINTERS_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Pointer to VS Sampler State** Format: DynamicStateOffset[31:5]SAMPLER_STATE*16 Specifies the 32-byte aligned address offset of the VS function's SAMPLER_STATE table. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_SBE

3DSTATE_SBE

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Fh 3DSTATE_SBE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 04h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**Reserved** Format: MBZ
	29	**Force Vertex URB Entry Read Length** Format: Enable This field provides a work around override for the computation of SBE_INT::Vertex URB Entry Read Length. If asserted, 3DSTATE_SBE::Vertex URB Entry Read Length is be used directly. Otherwise, SBE_INT::Vertex URB Entry Read Length is computed normally.
	28	**Force Vertex URB Entry Read Offset** Format: Enable This field provides a work around override for the computation of SBE_INT::Vertex URB Entry Read Offset. If asserted, 3DSTATE_SBE::Vertex URB Entry Read Offset is be used directly. Otherwise, SBE_INT::Vertex URB Entry Read Offset is computed normally.

3DSTATE_SBE

27:22 Number of SF Output Attributes

Format:	U6 count of attributes

Specifies the number of vertex attributes passed from the SF stage to the WM stage (does not include Position).

Value	Name
[0,32]	

21 Attribute Swizzle Enable

Format:	Enable

Enables the SF to perform swizzling on (up to the first 16) vertex attributes. If DISABLED, all vertex attributes are passed through.

20 Point Sprite Texture Coordinate Origin

This state controls how Point Sprite Texture Coordinates are generated (when enabled on a per-attribute basis by Point Sprite Texture Coordinate Enable).

Value	Name	Description
0h	UPPERLEFT	Top Left = (0,0,0,1)Bottom Left = (0,1,0,1)Bottom Right = (1,1,0,1)
1h	LOWERLEFT	Top Left = (0,1,0,1)Bottom Left = (0,0,0,1)Bottom Right = (1,0,0,1)

19 Primitive ID Override Component W

Format:	Enable

If set, the W component of output attribute selected by Primitive ID Override Attribute Select is overridden with the Primitive ID.

18 Primitive ID Override Component Z

Format:	Enable

If set, the Z component of output attribute selected by Primitive ID Override Attribute Select is overridden with the Primitive ID.

17 Primitive ID Override Component Y

Format:	Enable

If set, the Y component of output attribute selected by Primitive ID Override Attribute Select is overridden with the Primitive ID.

16 Primitive ID Override Component X

Format:	Enable

If set, the X component of output attribute selected by Primitive ID Override Attribute Select is overridden with the Primitive ID.

15:11 Vertex URB Entry Read Length

Format:	U5

Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments.

3DSTATE_SBE

Value	Name
[1,16]	

Programming Notes

It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read.This field should be set to the minimum length required to read the maximum source attribute. The maximum source attribute is indicated by the maximum value of the enabled Attribute # Source Attribute if Attribute Swizzle Enable is set, Number of Output Attributes-1 if enable is not set.
read_length = ceiling((max_source_attr+1)/2)

Vertex URB Entry Read Offset (10:5)

Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB.

Primitive ID Override Attribute Select (4:0)

Specifies which attribute is overridden w/ the Primitive ID

Programming Notes

Set all Primitive ID Override Component Select X/Y/Z/W to 0 to indicate there is no Primitive ID override.

Point Sprite Texture Coordinate Enable (2, 31:0)

Format:	Enable[32]

When processing point primitives, the attributes from the incoming point vertex are typically copied to the point object corner vertices. However, if a bit is set in this field, the corresponding Attribute is selected as a Point Sprite Texture Coordinate, in which case each corner vertex is assigned a pre-defined texture coordinate as defined by the Point Sprite Texture Coordinate Origin state bit. Bit 0 corresponds to output Attribute 0.

Constant Interpolation Enable (3, 31:0)

Format:	Enable[32]

This field is a bitmask containing a Constant Interpolation Enable bit for each corresponding attribute. If a bit is set, that attribute will undergo constant interpolation, and the corresponding WrapShortest Enable bits (if defined) will be ignored. If a bit is clear, components which are not enabled for WrapShortest interpolation (if defined) will be linearly interpolated.

Attribute 15 Active Component Format (4, 31:30)

Format:	Attribute_Component_Format

This state indicates which components of Attribute 15 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.

Attribute 14 Active Component Format (29:28)

Format:	Attribute_Component_Format

This state indicates which components of Attribute 14 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is

3DSTATE_SBE

		UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	27:26	**Attribute 13 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 13 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	25:24	**Attribute 12 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 12 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	23:22	**Attribute 11 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 11 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	21:20	**Attribute 10 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 10 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	19:18	**Attribute 9 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 9 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	17:16	**Attribute 8 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 8 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	15:14	**Attribute 7 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 7 are being used by the pixel shader Kernel.

3DSTATE_SBE		
		SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	13:12	**Attribute 6 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 6 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	11:10	**Attribute 5 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 5 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	9:8	**Attribute 4 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 4 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	7:6	**Attribute 3 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 3 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	5:4	**Attribute 2 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 2 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	3:2	**Attribute 1 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 1 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	1:0	**Attribute 0 Active Component Format** Format: **Attribute_Component_Format**

3DSTATE_SBE

		This state indicates which components of Attribute 0 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
5	31:30	**Attribute 31 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 31 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	29:28	**Attribute 30 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 30 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	27:26	**Attribute 29 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 29 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	25:24	**Attribute 28 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 28 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	23:22	**Attribute 27 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 27 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	21:20	**Attribute 26 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 26 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.

3DSTATE_SBE		
	19:18	**Attribute 25 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 25 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	17:16	**Attribute 24 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 24 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	15:14	**Attribute 23 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 23 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	13:12	**Attribute 22 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 22 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	11:10	**Attribute 21 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 21 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	9:8	**Attribute 20 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 20 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	7:6	**Attribute 19 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 19 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.

3DSTATE_SBE		
	5:4	**Attribute 18 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 18 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	3:2	**Attribute 17 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 17 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.
	1:0	**Attribute 16 Active Component Format** Format: **Attribute_Component_Format** This state indicates which components of Attribute 16 are being used by the pixel shader Kernel. SBE will not perform attribute delta calculations for any disabled components. Operation is UNDEFINED if kernel uses attribute vertex delta for any disabled component.

3DSTATE_SBE_SWIZ

3DSTATE_SBE_SWIZ

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 51h 3DSTATE_SBE_SWIZ Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n Total Length - 2
1..8	15:0	**Attribute** Format: SF_OUTPUT_ATTRIBUTE_DETAIL[16]
9..10	63:60	**Attribute 15 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	59:56	**Attribute 14 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	55:52	**Attribute 13 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	51:48	**Attribute 12 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE

3DSTATE_SBE_SWIZ		
	47:44	**Attribute 11 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	43:40	**Attribute 10 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	39:36	**Attribute 09 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	35:32	**Attribute 08 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	31:28	**Attribute 07 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	27:24	**Attribute 06 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	23:20	**Attribute 05 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	19:16	**Attribute 04 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	15:12	**Attribute 03 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	11:8	**Attribute 02 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	7:4	**Attribute 01 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	3:0	**Attribute 00 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE

3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_SCISSOR_STATE_POINTERS

Source: RenderCS

Length Bias: 2

The 3DSTATE_SCISSOR_STATE_POINTERS command is used to define the location of the indirect SCISSOR_RECT state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Fh 3DSTATE_SCISSOR_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**Scissor Rect Pointer** Format: DynamicStateOffset[31:5]SCISSOR_RECT*16 Specifies the 32-byte aligned address offset of the SCISSOR_RECT state. This offset is relative to the **Dynamic State Base Address**.
	4:0	**Reserved** Format: MBZ

3DSTATE_SF

3DSTATE_SF

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 13h 3DSTATE_SF Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**Reserved** Format: MBZ
	29:12	**Line Width** Format: U11.7 Range: [0.0, 2047.9921875] Controls width of line primitives. Setting a Line Width of 0.0 specifies the rasterization of the "thinnest" (one-pixel-wide), non-antialiased lines. Note that this effectively overrides the effect of AAEnable (though the AAEnable state variable is not modified). **Programming Notes** Software must not program a value of 0.0 when running in MSRASTMODE_ON_xxx modes -

3DSTATE_SF

		zero-width lines are not available when multisampling rasterization is enabled. **Restriction** Line widths higher than 40.0 will appear to be thinner than the programmed line width at certain angles. Software must expose the line widths higher than 40 to APIs only if the loss in quality is acceptable. Software must set the "WM_CHIKEN1" register bits "1 and 3" when rendering the AA lines higher than Line width of "40.0".
	11	**Legacy Global Depth Bias Enable** Format: Enable Enables the SF to use the Global Depth Offset Constant state unmodified. If this bit is not set, the SF will scale the Global Depth Offset Constant as described in section Error! Reference source not found. of this document. **Programming Notes** This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.
	10	**Statistics Enable** Format: Enable If ENABLED, this FF unit will increment CL_PRIMITIVES_COUNT on behalf of the CLIP stage. If DISABLED, CL_PRIMITIVES_COUNT will be left unchanged. **Programming Notes** This bit should be set whenever clipping is enabled and the Statistics Enable bit is set in CLIP_STATE. It should be cleared if clipping is disabled or Statistics Enable in CLIP_STATE is clear.
	9:2	**Reserved** Format: MBZ
	1	**Viewport Transform Enable** Format: Enable This bit controls the Viewport Transform function.
	0	**Reserved** Format: MBZ
2	31:29	**Reserved** Format: MBZ
	28	**Reserved**
	27:18	**Reserved** Format: MBZ
	17:16	**Line End Cap Antialiasing Region Width**

3DSTATE_SF

		Format: U2 This field specifies the distances over which the coverage of anti-aliased line end caps are computed. **Value** / **Name** / **Description** 0h / 0.5 pixels / 0.5 pixels 1h / 1.0 pixels / 1.0 pixels 2h / 2.0 pixels / 2.0 pixels 3h / 4.0 pixels / 4.0 pixels
	15	**Reserved** Format: MBZ
	14	**Reserved** Format: MBZ
	13	**Reserved**
	12	**Reserved**
	11:0	**Reserved** Format: MBZ
3	31	**Last Pixel Enable** Format: Enable If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines). **Programming Notes** Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESTRIP.
	30:29	**Triangle Strip/List Provoking Vertex Select** Format: U2 Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives.Does current implementation send provoking vertex first? **Value** / **Name** 0h / 0 1h / 1 2h / 2 3h / Reserved
	28:27	**Line Strip/List Provoking Vertex Select** Format: U2 Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex". **Value** / **Name** / **Description** 0h / 0 / Vertex 0

3DSTATE_SF

1h	1	Vertex 1
2h	Reserved	Reserved
3h	Reserved	Reserved

26:25 **Triangle Fan Provoking Vertex Select**

Format:	U2

Selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".

Value	Name
0h	0
1h	1
2h	2
3h	Reserved

24:15 **Reserved**

Format:	MBZ

14 **AA Line Distance Mode**

Format:	U1

This bit controls the distance computation for antialiased lines.

Value	Name	Description
1h	AALINEDISTANCE_TRUE	True distance computation. This is the normal setting which should yield WHQL compliance.

13 **Smooth Point Enable**

Format:	Enable
Double Buffer Armed By:	Enables logic to draw smooth OGL Points

Programming Notes
If Enabled, SF will treat points in the same fashion that AA lines are processed

12 **Vertex Sub Pixel Precision Select**

Format:	U1

Selects the number of fractional bits maintained in the vertex data

Value	Name	Description
0h	8	8 sub pixel precision bits maintained
1h	4	4 sub pixel precision bits maintained

Programming Notes
When Conservative Rasterization is enabled, this bit must be programmed to 0.

11 **Point Width Source**

Controls whether the point width passed on the vertex or from state is used for rendering point

3DSTATE_SF

primitives.

Value	Name	Description
0h	Vertex	Use Point Width on Vertex
1h	State **[Default]**	Use Point Width from State

10:0 **Point Width**

Format:	U8.3

Range: [0.125, 255.875] pixels
This field specifies the size (width) of point primitives in pixels. This field is overridden (though not overwritten) whenever point width information is passed in the FVF

3DSTATE_SO_BUFFER

3DSTATE_SO_BUFFER

Source: RenderCS

Length Bias: 2

Programming Notes

Foreach SO Buffer, the 3DSTATE_SO_BUFFER must only be sent once prior to each 3DPRIMITIVE command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 18h 3DSTATE_SO_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**SO Buffer Enable** Format: Enable If set, stream output to SO Buffer is enabled, , if 3DSTATE_STREAMOUT::SO Function ENABLE is also enabled..If clear, the SO Buffer is considered "not bound" and effectively treated as a zero-length buffer for the purposes of SO output and overflow detection. If an enabled stream's Stream to Buffer Selects includes this buffer it is by definition an overflow condition. That stream will cause no writes to occur, and only SO_PRIM_STORAGE_NEEDED[<stream>] will increment.

3DSTATE_SO_BUFFER

	30:29	**SO Buffer Index** Format: U2 Specifies which of the four SO Buffers is being defined.
	28:22	**SO Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the SO buffer.
	21	**Stream Offset Write Enable** Format: Enable When set, this field allows the hardware to write SO_WRITE_OFFSET[Buffer#] as specified in the Stream Offset field. **Programming Notes** The field is operates irrespective of whether SO Buffer Enable is set or clear.
	20	**Stream Output Buffer Offset Address Enable** Format: Enable When set, this field allows the hardware to read/write the stream output buffer offset as specified in the "Stream Output Buffer Offset Address" field. **Programming Notes** The field is operates irrespective of whether SO Buffer Enable is set or clear.
	19:12	**Reserved** Format: MBZ
	11:0	**Reserved** Format: MBZ
2..3	63:2	**Surface Base Address** Format: GraphicsAddress63-2 This field specifies the starting address of the buffer in Graphics Memory.
	1:0	**Reserved** Format: MBZ
4	31:30	**Reserved** Format: MBZ
	29:0	**Surface Size** Format: U30-1 This field specifies the size of buffer in number DWords minus 1 of the buffer in Graphics Memory.

3DSTATE_SO_BUFFER		
5..6	63:2	**Stream Output Buffer Offset Address** Format: GraphicsAddress63-2 This field specifies the starting address of the buffer in Graphics Memory where the Stream Output Buffer Offset is stored when all the data has been written. It is also used to fetch the stream Output buffer Offset when needed.
	1:0	**Reserved** Format: MBZ
7	31:0	**Stream Offset** This field specifies the Offset in stream output buffer to start at, or whether to append to the end of an existing buffer. The Offset must be DWORD aligned. If Stream Offset is equal to 0xFFFFFFFF then load the value at the Stream Output Buffer Offset address into SO_WRITE_OFFSET[Buffer#]. Otherwise, SO_WRITE_OFFSET[Buffer#] = Stream Offset.

3DSTATE_SO_DECL_LIST

3DSTATE_SO_DECL_LIST

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 17h 3DSTATE_SO_DECL_LIST Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Total Length - 2 Value | Name | Description [1,257] | Excludes DWORD (0,1) 0-128 Entries | Value = 2 * (# of SO_DECL quads) + 1
1	31:16	**Reserved** Format: MBZ
	15:12	**Stream to Buffer Selects [3]** Format: U4 bitmask Index of SO Stream Identifies to which SO Buffers stream 3 outputs. See Stream To Buffer Selects [0] field description. Value | Name 1xxxb | SO Buffer 3 x1xxb | SO Buffer 2 xx1xb | SO Buffer 1 xxx1b | SO Buffer 0

3DSTATE_SO_DECL_LIST

	11:8	**Stream to Buffer Selects [2]** Format: U4 bitmask Identifies to which SO Buffers stream 2 outputs. See Stream To Buffer Selects [0] field description. **Value** / **Name** 1xxxb — SO Buffer 3 x1xxb — SO Buffer 2 xx1xb — SO Buffer 1 xxx1b — SO Buffer 0
	7:4	**Stream to Buffer Selects [1]** Format: U4 bitmask Identifies to which SO Buffers stream 1 outputs. See Stream To Buffer Selects [0] field description. **Value** / **Name** 1xxxb — SO Buffer 3 x1xxb — SO Buffer 2 xx1xb — SO Buffer 1 xxx1b — SO Buffer 0
	3:0	**Stream to Buffer Selects [0]** Format: U4 bitmask Identifies to which SO Buffers stream 0 outputs (irrespective of whether those buffers are enabled via 3DSTATE_STREAMOUT). Software is required to scan the SO_DECL list in order to provide this summary information. Note: For "inactive" streams, software must program this field to all zero (no buffers written to) and the corresponding Num Entries field to zero (no valid SO_DECLs). **Value** / **Name** 1xxxb — SO Buffer 3 x1xxb — SO Buffer 2 xx1xb — SO Buffer 1 xxx1b — SO Buffer 0
2	31:24	**Num Entries [3]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 3. (See notes in Num Entries [0] field description). **Value** / **Name** [0,128] — entries

3DSTATE_SO_DECL_LIST

	23:16	**Num Entries [2]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 2. (See notes in Num Entries [0] field description). **Value** [0,128] — **Name** entries
	15:8	**Num Entries [1]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 1. (See notes in Num Entries [0] field description). **Value** [0,128] — **Name** entries
	7:0	**Num Entries [0]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 0.Note that the SO_DECLs are programmed in groups of four (one SO_DECL for each of the four streams). Therefore the number of 2-DWord groups of SO_DECLs supplied in this command is derived from the stream(s) with the most valid SO_DECLs. The NumEntries value specific to each stream will indicate how many SO_DECLS are valid for that particular stream. Any trailing invalid SO_DECLs supplied for streams with fewer valid SO_DECLs will be ignored. It is legal to specify Num Entries = 0 for all four streams simultaneously. In this case there will be no SO_DECLs included in the command (only DW 0-2). Note that all Stream to Buffer Selects bits must be zero in this case (as no streams produce output). **Value** [0,128] — **Name** entries
3..n	63:0	**Entry** Format: SO_DECL_ENTRY

3DSTATE_STENCIL_BUFFER

3DSTATE_STENCIL_BUFFER

Source: RenderCS

Length Bias: 2

Description

This command sets the surface state of the separate stencil buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

Programming Notes

Restriction: Prior to changing Depth/Stencil Buffer state (i.e., any combination of 3DSTATE_DEPTH_BUFFER, 3DSTATE_CLEAR_PARAMS, 3DSTATE_STENCIL_BUFFER, 3DSTATE_HIER_DEPTH_BUFFER) SW must first issue a pipelined depth stall (PIPE_CONTROL with Depth Stall bit set, followed by a pipelined depth cache flush (PIPE_CONTROL with Depth Flush Bit set, followed by another pipelined depth stall (PIPE_CONTROL with Depth Stall Bit set), unless SW can otherwise guarantee that the pipeline from WM onwards is already flushed (e.g., via a preceding MI_FLUSH).

The stencil buffer is always Tile-W

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 06h 3DSTATE_STENCIL_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Format: =n Total Length - 2 Value: 3h — Name: Excludes Dword (0,1) **[Default]**

3DSTATE_STENCIL_BUFFER

1	31	**Stencil Buffer Enable** Format: U1 When set indicates that there is a valid stencil buffer. **Programming Notes** This bit should be "0" if Depth buffer surface format is D16_UNORM OR Depth buffer surface type is NULL.
	30:29	**Reserved** Format: MBZ
	28:22	**Stencil Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the stencil buffer.
	21	**Reserved** Format: MBZ
	20:17	**Reserved** Format: MBZ
	16:0	**Surface Pitch** Format: U17-1 Pitch in Bytes This field specifies the pitch of the stencil buffer in (#Bytes - 1). **Value** / **Name** / **Description** [127, 1FFFFh] / / corresponding to [128B, 128KB]also restricted to a multiple of 128B **Programming Notes** Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB]. The pitch must be set to 2x the value computed based on width, as the stencil buffer is stored with two rows interleaved. For details on the separate stencil buffer storage format in memory, see GPU Overview (vol1a), Memory Data Formats, Surface Layout, 2D Surfaces, Stencil Buffer Layout (section 8.20.4.8).
2..3	63:0	**Surface Base Address** Format: GraphicsAddress[63:0]Stencil_Buffer This field specifies the address of the buffer in Graphics Memory. **Programming Notes** The Stencil Buffer can only be mapped to Main Memory (uncached).
4	31:15	**Reserved** Format: MBZ
	14:0	**Surface QPitch** Format: QPitch[16:2]

3DSTATE_STENCIL_BUFFER

Description
This field specifies the distance in rows between array slices. It is used only in the following cases: • **Surface Array** is enabled *OR* • **Number of Mulitsamples** is not NUMSAMPLES_1 and **Multisampled Surface Storage Format** set to MSFMT_MSS *OR* • **Surface Type** is SURFTYPE_CUBE
The interpretation of this field is dependent on **Surface Type** as follows: • SURFTYPE_1D: distance in *pixels* between array slices • SURFTYPE_2D/CUBE: distance in *rows* between array slices • SURFTYPE_3D: distance in *rows* between R-slices

Value	Name	Description
[4h,1FFFCh]		in multiples of 4 (low 2 bits missing)

Programming Notes
This field must be set to an integer multiple of 8 (QPitch[2] MBZ) Software must ensure that this field is set to a value sufficiently large such that the array slices in the surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored in memory.

3DSTATE_STREAMOUT

3DSTATE_STREAMOUT

Source: RenderCS

Length Bias: 2

This command contains pipelined state required by the SOL unit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Eh 3DSTATE_STREAMOUT Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**SO Function Enable** Format: U1 If set, the SO function is enabled. Vertex data will be streamed out to memory (subject to overflow detection) as controlled by the various SO-related state variables. If clear, the SO function is disabled, and therefore no vertex data will be streamed out to memory. However, the Rendering Disable and Render Stream Select fields will still be used to determine which vertices (if any) are forwarded down the pipeline for (possible) rendering.
	30	**API Rendering Disable** Format: U1 If set, Indicates the API wants the SO stage not to forward any topologies down the pipeline. If clear, Indicates the API wants the SO stage to forward topologies associated with **Render Stream Select** down the pipeline. This bit is used even if **SO Function Enable** is DISABLED.

3DSTATE_STREAMOUT

		Programming Notes The SOL unit generates an SOL_INT::Render_Enable which ultimately controls whether rendering occurs or not.
	29	**Reserved** Format: MBZ
	28:27	**Render Stream Select** Format: U2 **Description** This field specifies which stream has been selected to be forwarded down the pipeline for possible rendering. Topologies from other streams will not be passed down the pipeline. If Rendering Disable is set, this field is ignored, as no topologies are sent down the pipeline. SO Function Enable must also be ENABLED in order for thiis field to select a stream for rendering. When **SO Function Enable** is DISABLED and Rendering Disable is cleared (i.e., rendering is enabled), StreamID is ignored downstream of the SO stage, allowing any stream to be rendered.
	26	**Reorder Mode** This bit controls how vertices of triangle objects in TRISTRIP[_ADJ] and TRISTRIP_REV are reordered for the purposes of stream-out only (does not impact rendering). See table in Input Buffering. **Value** / **Name** / **Description** 0h / LEADING / Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the leading (first) vertices are in consecutive order starting at v0. A similar reordering is performed on alternating triangles in a TRISTRIP_REV. 1h / TRAILING / Reorder the vertices of alternating triangles of a TRISTRIP[_ADJ] such that the trailing (last) vertices are in consecutive order starting at v2. A similar reordering is performed on alternating triangles in a TRISTRIP_REV.
	25	**SO Statistics Enable** Format: Enable This bit controls whether StreamOutput statistics register(s) can be incremented. **Value** / **Name** / **Description** 0h / Disable / SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers cannot increment. 1h / Enable / SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers can increment.

3DSTATE_STREAMOUT

	24:23	**Force Rendering** This field provides a work around override for the computation of SOL_INT::Render_Enable **Value** / **Name** / **Description** 0h / Normal / SOL_INT::Render_Enable is computed normally 1h / Resreved / 2h / Force_Off / Forces the rendering to be disabled. 3h / Force_on / Forces the rendering to be enabled.
	22:21	**Reserved** Format: MBZ
	20:12	**Reserved** Format: MBZ
	11:8	**Reserved** Format: MBZ
	7:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ
	29	**Stream 3 Vertex Read Offset** Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 3 vertex data. (See **Stream 0 Vertex Read Offset**)
	28:24	**Stream 3 Vertex Read Length** Format: U5-1 count of 256-bit units (See Stream 0 Vertex Read Length)
	23:22	**Reserved** Format: MBZ
	21	**Stream 2 Vertex Read Offset** Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 2 vertex data. (See Stream 0 Vertex Read Offset)
	20:16	**Stream 2 Vertex Read Length** Format: U5-1 count of 256-bit units
	15:14	**Reserved** Format: MBZ

3DSTATE_STREAMOUT

	13	**Stream 1 Vertex Read Offset** Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 1 vertex data. (See Stream 0 Vertex Read Offset)
	12:8	**Stream 1 Vertex Read Length** Format: U5-1 count of 256-bit units (See Stream 0 Vertex Read Length)
	7:6	**Reserved** Format: MBZ
	5	**Stream 0 Vertex Read Offset** Format: U1 count of 256-bit units Specifies amount of data to skip over before reading back Stream 0 vertex data. Must be zero if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).
	4:0	**Stream 0 Vertex Read Length** Format: U5-1 count of 256-bit units Specifies amount of vertex data to read back for Stream 0 vertices, starting at the Stream 0 Vertex Read Offset location. Maximum readback is 17 256-bit units (34 128-bit vertex attributes). Read data past the end of the valid vertex data has undefined contents, and therefore shouldn't be used to source stream out data. Must be zero (i.e., read length = 256b) if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).
3	31:28	**Reserved** Format: MBZ
	27:16	**Buffer 1 Surface Pitch**
	15:12	**Reserved** Format: MBZ
	11:0	**Buffer 0 Surface Pitch** Format: U12 pitch in Bytes This field specifies the pitch of the SO buffer in #Bytes. **Value** / **Name** [0,2048] / Must be 0 or a multiple of 4 Bytes. **Programming Notes**

3DSTATE_STREAMOUT		
		A Surface Pitch of 0 indicates an un-bound buffer. No writes are performed. Surface Base Address is ignored.
4	31:28	**Reserved** Format: MBZ
	27:16	**Buffer 3 Surface Pitch** Format: U12
	15:12	**Reserved** Format: MBZ
	11:0	**Buffer 2 Surface Pitch** Format: U12

3DSTATE_TE

3DSTATE_TE

Source: RenderCS

Length Bias: 2

The state used by TE is defined with this inline state packet.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_TE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 2h — Name: Excludes DWord (0,1) **[Default]**
1	31:24	**Reserved** Format: MBZ
	23:22	**Reserved** Format: MBZ
	21	**Reserved** Format: MBZ
	20	**Reserved** Format: MBZ
	19	**Reserved** Format: MBZ

3DSTATE_TE

18:17 Reserved

Format:	MBZ

16 Reserved

Format:	MBZ

15:14 Reserved

Format:	MBZ

13:12 Partitioning

Format:	U2

This field specifies how edges are partitioned based on tessellation factor.

Value	Name	Description
0h	INTEGER	Outside/inside edges are divided into an integer number of equal-sized segments.
1h	ODD_FRACTIONAL	Outside/inside edges are divided into an odd number of possibly-unequal-sized segments.
2h	EVEN_FRACTIONAL	Outside/inside edges are divided into an even number of possibly-unequal-sized segments.

11:10 Reserved

Format:	MBZ

9:8 Output Topology

Format:	U2

This field specifies which primitive types are to be output.

Value	Name	Description
0h	POINT	Points are output (as POINTLIST topologies)
1h	LINE	Lines are output (as LINESTRIP topologies). Only valid if ISOLINE domain is selected.
2h	TRI_CW	Clockwise-ordered triangles are output (either as TRISTRIP, TRISTRIP_REV or TRILIST topologies). Not valid if ISOLINE domain is selected.
3h	TRI_CCW	Count-clockwise-ordered triangles are output (either as TRISTRIP, TRISTRIP_REV or TRILIST topologies). Not valid if ISOLINE domain is selected.

7:6 Reserved

Format:	MBZ

5:4 TE Domain

Format:	U2

This field specifies which type of domain is to be tessellated.

Value	Name	Description
0h	QUAD	2D (U, V) domain is tessellated
1h	TRI	Triangular (U, V, W) domain is tessellated

3DSTATE_TE

		2h	ISOLINE	2D (U, V) domain is tessellated.

3 — Reserved

Format:	MBZ

2:1 — TE Mode

Format:	U2

When TE Enable is ENABLED, this field specifies the overall operation of the TE stage.This field is ignored if TE Enable is DISABLED.

Value	Name	Description
0h	HW_TESS	Normal HW Tessellation Mode. The TessFactors are read from the patch URB entry, and are used to perform fixed-function hardware tessellation of the specified domain.

0 — TE Enable

Format:	Enable

If ENABLED, the TE stage will perform tessellation processing on incoming patch primitives. The TE Mode field determines how this tessellation operation proceeds.If DISABLED, the TE goes into pass-through mode. All other state fields are ignored.

Programming Notes
The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.

2 — 31:0 — Maximum Tessellation Factor Odd

Format:	IEEE_Float

This field specifies the maximum TessFactor for ODD_FRACTIONAL partitioning when in HW_TESS mode.

Value	Name	Description
[427c0000h,427c0000h]	63 **[Default]**	Per API Spec, For normal operation software should set this value to 63.0

Programming Notes
Note that ISOLINE's LineDensity TF is always subjected to INTEGER partitioning regardless of the Partitioning state.

3 — 31:0 — Maximum Tessellation Factor Not Odd

Format:	IEEE_Float

This field specifies the maximum TessFactor for EVEN_FRACTIONAL or INTEGER partitioning when in HW_TESS mode.

Value	Name	Description
[42800000h,42800000h]	64 **[Default]**	Per API Spec, For normal operation software should set this value to 64.0

3DSTATE_TE		
		Programming Notes
		Note that ISOLINE's LineDensity TF is always subjected to INTEGER partitioning regardless of the Partitioning state.

3DSTATE_URB_CLEAR

3DSTATE_URB_CLEAR

Source: RenderCS

Length Bias: 2

The 3DSTATE_URB_CLEAR command allows SW to clear (write zero) to a section in the URB.

Programming Notes

- The command temporarily halts command execution.
- This command is not a part of context save/restore.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Dh 3DSTATE_URB_CLEAR Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**Reserved** Format: MBZ
	29:16	**URB Clear Length** This field specifies the number of 256b entries in the URB to be cleared to zero.
	15	**Reserved** Format: MBZ

3DSTATE_URB_CLEAR		
	14:0	**URB Address** Format: URBAddress[19:5] 256b aligned This field specifies Bits 19:5 of the URB Address

3DSTATE_URB_DS

3DSTATE_URB_DS

Source: RenderCS

Length Bias: 2

Description

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

When programming DS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 32h 3DSTATE_URB_DS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n

3DSTATE_URB_DS

1 — 31:25 — DS URB Starting Address

Format:	U7

Offset from the start of the URB memory where DS starts its allocation, specified in multiples of 8 KB.

Value	Name	Exists If
[0,48]		Device[SliceCount] == 1
[4,48]		Device[SliceCount] GT 1

24:16 — DS URB Entry Allocation Size

Format:	U9-1 Count of 512-bit units

Specifies the length of each URB entry owned by DS. This field is always used (even if DS Function Enable is DISABLED).

Value	Name
[0,9]	

15:0 — DS Number of URB Entries

Description
Specifies the number of URB entries that are used by DS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below. This field is always used (even if DS Function Enable is DISABLED).
If Domain Shader Thread Dispatch is Enabled then the minimum number of handles that must be allocated is 34 URB entries.

Value	Name
[0,1120]	

Programming Notes
DS Number of URB Entries must be divisible by 8 if the DS URB Entry Allocation Size is programmed to a value less than 9, which is 10 512-bit URB entries. "2:0" = reserved "000"

3DSTATE_URB_GS

3DSTATE_URB_GS

Source: RenderCS

Length Bias: 2

Description

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations

Programming Notes

When programming GS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_DS must also be programmed in order for the programming of this state to be valid.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 33h 3DSTATE_URB_GS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n

3DSTATE_URB_GS

1 — 31:25 — GS URB Starting Address

Format:	U7

Offset from the start of the URB memory where GS starts its allocation, specified in multiples of 8 KB.

Value	Name	Exists If
[0,48]		Device[SliceCount] == 1
[4,48]		Device[SliceCount] GT 1

24:16 — GS URB Entry Allocation Size

Format:	U9-1 512-bit units

Specifies the length of each URB entry owned by GS. This field is always used (even if GS Function Enable is DISABLED).

15:0 — GS Number of URB Entries

Specifies the number of URB entries that are used by GS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below.
This field is always used (even if GS Function Enable is DISABLED).

Programming Notes
Only if GS is disabled can this field be programmed to 0. If GS is enabled this field shall be programmed to a value greater than 0. For GS Dispatch Mode "Single", this field shall be programmed to a value greater than or equal to 1. For other GS Dispatch Modes, refer to the definition of Dispatch Mode (3DSTATE_GS) for minimum values of this field.
GS Number of URB Entries must be divisible by 8 if the GS URB Entry Allocation Size is less than 9 512-bit URB entries. "2:0" = reserved "000"
When 3DSTATE_GS:Enable is true, the minimum number of GS Number of URB Entries must be set to 2.

3DSTATE_URB_HS

3DSTATE_URB_HS

Source: RenderCS

Length Bias: 2

Description

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations

Programming Notes

When programming HS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 31h 3DSTATE_URB_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:25	**HS URB Starting Address** Format: U7

3DSTATE_URB_HS

Offset from the start of the URB memory where HS starts its allocation, specified in multiples of 8 KB.

Value	Name	Exists If
[0,48]		Device[SliceCount] == 1
[4,48]		Device[SliceCount] GT 1

24:16 **HS URB Entry Allocation Size**

Format:	U9-1 Count of 512-bit units

Specifies the length of each URB entry owned by HS. This field is always used (even if HS Function Enable is DISABLED).

15:0 **HS Number of URB Entries**

Specifies the number of URB entries that are used by HS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below.
This field is always used (even if HS Function Enable is DISABLED).
Programming Restriction:HS Number of URB Entries must be divisible by 8 if the HS URB Entry Allocation Size is less than 9 512-bit URB entries."2:0" = reserved "000"

Value	Name
[0,672]	
[0,128]	

Programming Notes
When 3DSTATE_HS:Enable is true and 3DSTATE_HS:Dispatch Mode is set to 8 patch, the minimum number of HS Number of URB Entries must be set to 16. When 3DSTATE_HS:Enable is true and 3DSTATE_HS:Dispatch Mode is set to dual patch, the minimum number of HS Number of URB Entries must be set to 4.

3DSTATE_URB_VS

3DSTATE_URB_VS

Source: RenderCS, PositionCS

Length Bias: 2

Description

VS URB Entry Allocation Size equal to 4(5 512-bit URB rows) may cause performance to decrease due to banking in the URB. Element sizes of 16 to 20 should be programmed with six 512-bit URB rows.

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The offset and size should be programmed as if there is only one slice enabled. Hardware will grow the size based on the slice configuration. Software shall ensure that the values programmed do not exceed the URB capacity of one slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

When programming VS URB state for the RCS 3D pipe, 3DSTATE_URB_HS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 30h 3DSTATE_URB_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:25	**VS URB Starting Address** Format: U7

3DSTATE_URB_VS

Offset from the start of the URB memory where VS starts its allocation, specified in multiples of 8 KB.

Value	Name	Exists If
[0,48]		Device[SliceCount] == 1
[4,48]		Device[SliceCount] GT 1

24:16 **VS URB Entry Allocation Size**

Format:	U9-1 count of 512-bit units

Specifies the length of each URB entry owned by VS. This field is always used (even if VS Function Enable is DISABLED).

Programming Notes
Programming Restriction: As the VS URB entry serves as both the per-vertex input and output of the VS shader, the VS URB Allocation Size must be sized to the maximum of the vertex input and output structures.

15:0 **VS Number of URB Entries**

Format:	U16

Specifies the number of URB entries that are used by VS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below.
This field is always used (even if VS Function Enable is DISABLED).

Value	Name
[64,1856]	
[34,352]	

Programming Notes
Programming Restriction: VS Number of URB Entries must be divisible by 8 if the VS URB Entry Allocation Size is less than 9 512-bit URB entries."2:0" = reserved "000b"

3DSTATE_VERTEX_BUFFERS

3DSTATE_VERTEX_BUFFERS

Source: RenderCS

Length Bias: 2

Description

This command is used to specify VB state used by the VF function.

: Can specify from 1 to 33 VBs.

The VertexBufferID field within a VERTEX_BUFFER_STATE structure indicates the specific VB. If a VB definition is not included in this command, its associated state is left unchanged and is available for use if previously defined.

Programming Notes

It is possible to have individual vertex elements sourced completely from generated ID values and therefore not require any vertex buffer accesses for that vertex element. In this case, VF function will simply ignore the VB state associated with that vertex element. If all enabled vertex elements have this characteristic, no VBs are required to process 3DPRIMITIVE commands. For example, this might arise when the user wants to perform all data lookups in the first shader, so only generated index values need to be passed down to it. In this extreme case, SW would not need to program any VB state, and therefore not need to issue any 3DSTATE_VERTEX_BUFFERS commands.

For any 3DSTATE_VERTEX_BUFFERS command, at least one VERTEX_BUFFER_STATE structure must be included.

VERTEX_BUFFER_STATE structures are 4 DWords for both VERTEXDATA buffers and INSTANCEDATA buffers.

Inclusion of partial VERTEX_BUFFER_STATE structures is UNDEFINED.

The order in which VBs are defined within this command can be arbitrary, though a vertex buffer must be defined only once in any given command (otherwise operation is UNDEFINED).

The VF cache (and the VFR cache if POSH is enabled) needs to be invalidated before binding and then using Vertex Buffers that overlap with any previously bound Vertex Buffer (at a 64B granularity) since the last invalidation.
A VF cache invalidate is performed by setting the "VF Cache Invalidation Enable" bit in PIPE_CONTROL.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode

3DSTATE_VERTEX_BUFFERS

	23:16	**3D Command Sub Opcode** Default Value: 08h 3DSTATE_VERTEX_BUFFERS Format: Opcode
	15:8	**Reserved**
	7:0	**DWord Length** Format: =n n = 4b-1 (where b = # of buffer states included) Value: 3 — Name: DWORD_COUNT_n **[Default]** Value: [3,131] — Name: 1-33 Buffers
1..n	127:0	**Vertex Buffer State** Format: VERTEX_BUFFER_STATE

3DSTATE_VERTEX_ELEMENTS

3DSTATE_VERTEX_ELEMENTS

Source: RenderCS

Length Bias: 2

Description
This is a variable-length command used to specify the active vertex elements. Each VERTEX_ELEMENT_STATE structure contains a Valid bit which determines which elements are used.
Up to 34 elements.

Programming Notes
At least one VERTEX_ELEMENT_STATE structure must be included.
Inclusion of partial VERTEX_ELEMENT_STATE structures is UNDEFINED.
SW must ensure that at least one vertex element is defined prior to issuing a 3DPRIMTIVE command, or operation is UNDEFINED.
There are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.
Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.
See additional restrictions listed in the command fields and VERTEX_ELEMENT_STATE description.
Element[0] must be valid.
All elements must be valid from Element[0] to the last valid element. (I.e. if Element[2] is valid then Element[1] and Element[0] must also be valid).
The pitch between elements packed in the URB will always be 128 bits.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h 3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode

3DSTATE_VERTEX_ELEMENTS

	23:16	**3D Command Sub Opcode** Default Value: 09h 3DSTATE_VERTEX_ELEMENTS Format: Opcode
	15:8	**Reserved**
	7:0	**DWord Length** Format: =n Vertex Element Count = (DWord Count + 1) / 2 **Value** / **Name** / **Description** 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,67] / Range / 1-34 Elements
1..n	63:0	**Element** Format: VERTEX_ELEMENT_STATE

3DSTATE_VF_COMPONENT_PACKING

3DSTATE_VF_COMPONENT_PACKING

Source:	RenderCS
Length Bias:	2

This command is used to specify, separately for Vertex Elements [0-31], which post-conversion, 32-bit components are "enabled" to be stored in the URB , and which are "disabled" (not stored). 128 per-component enable bits are provided. Disabling all four components for a given Vertex Element will result in no data stored for that element. Note that any insertion of SGVs (3DSTATE_VF_SGVS) is performed before the packing operation.

The **Component Packing Enable** bit (3DSTATE_VF) controls the overall packing process. If that bit is set, the packing process is enabled and the bit mask provided in this command is used to control which components are stored. If that bit is clear, the packing process is disabled - all four components of "valid" Vertex Elements will be stored.

Programming Notes

Programming Restrictions:

- The Vertex Elements referenced in this command correspond to the first 32 VERTEX_ELEMENT structures passed in 3DSTATE_VERTEX_ELEMENTS. A Vertex Element must be marked as "Valid" via 3DSTATE_VERTEX_ELEMENTS or be an SGV or an element between the last valid element and the last SGV in order for the corresponding Component Enable bits of this command to be utilized.
- No enable bits are provided for Vertex Elements [32-33], and therefore no packing is performed on these elements (if Valid, all 4 components are stored).
- If a Vertex Element has Edge Flag Enable set no packing is performed for that element and the corresponding packing state is ignored.
- Component packing is probably only useful for SIMD8 VS thread execution.

At least one component of one "valid" Vertex Element must be enabled.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_VF_COMPONENT_PACKING

	23:16	**3D Command Sub Opcode** Default Value: 55h 3DSTATE_VF_COMPONENT_PACKING Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:28	**Vertex Element 07 Enables** Format: COMPONENT_ENABLES
	27:24	**Vertex Element 06 Enables** Format: COMPONENT_ENABLES
	23:20	**Vertex Element 05 Enables** Format: COMPONENT_ENABLES
	19:16	**Vertex Element 04 Enables** Format: COMPONENT_ENABLES
	15:12	**Vertex Element 03 Enables** Format: COMPONENT_ENABLES
	11:8	**Vertex Element 02 Enables** Format: COMPONENT_ENABLES
	7:4	**Vertex Element 01 Enables** Format: COMPONENT_ENABLES
	3:0	**Vertex Element 00 Enables** Format: COMPONENT_ENABLES
2	31:28	**Vertex Element 15 Enables** Format: COMPONENT_ENABLES
	27:24	**Vertex Element 14 Enables** Format: COMPONENT_ENABLES
	23:20	**Vertex Element 13 Enables** Format: COMPONENT_ENABLES
	19:16	**Vertex Element 12 Enables** Format: COMPONENT_ENABLES
	15:12	**Vertex Element 11 Enables** Format: COMPONENT_ENABLES

3DSTATE_VF_COMPONENT_PACKING		
	11:8	**Vertex Element 10 Enables** Format: COMPONENT_ENABLES
	7:4	**Vertex Element 09 Enables** Format: COMPONENT_ENABLES
	3:0	**Vertex Element 08 Enables** Format: COMPONENT_ENABLES
3	31:28	**Vertex Element 23 Enables** Format: COMPONENT_ENABLES
	27:24	**Vertex Element 22 Enables** Format: COMPONENT_ENABLES
	23:20	**Vertex Element 21 Enables** Format: COMPONENT_ENABLES
	19:16	**Vertex Element 20 Enables** Format: COMPONENT_ENABLES
	15:12	**Vertex Element 19 Enables** Format: COMPONENT_ENABLES
	11:8	**Vertex Element 18 Enables** Format: COMPONENT_ENABLES
	7:4	**Vertex Element 17 Enables** Format: COMPONENT_ENABLES
	3:0	**Vertex Element 16 Enables** Format: COMPONENT_ENABLES
4	31:28	**Vertex Element 31 Enables** Format: COMPONENT_ENABLES
	27:24	**Vertex Element 30 Enables** Format: COMPONENT_ENABLES
	23:20	**Vertex Element 29 Enables** Format: COMPONENT_ENABLES
	19:16	**Vertex Element 28 Enables** Format: COMPONENT_ENABLES
	15:12	**Vertex Element 27 Enables** Format: COMPONENT_ENABLES
	11:8	**Vertex Element 26 Enables** Format: COMPONENT_ENABLES

3DSTATE_VF_COMPONENT_PACKING		
	7:4	**Vertex Element 25 Enables** Format: COMPONENT_ENABLES
	3:0	**Vertex Element 24 Enables** Format: COMPONENT_ENABLES

3DSTATE_VF

3DSTATE_VF

Source: RenderCS

Length Bias: 2

Description
This command is used to set various state variables in the VF stage.
The use of the component packing mask is specified via 3DSTATE_VF_COMPONENT_PACKING

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ch 3DSTATE_VF Format: OpCode
	15	**Reserved** Format: MBZ
	14	**Reserved** Format: MBZ
	13	**Reserved** Format: MBZ
	12	**Reserved**
	11	**Reserved** Format: MBZ
	10	**Sequential Draw Cut Index Enable** Format: Enable If ENABLED, vertex indices in SEQUENTIAL 3DPRIMITIVE commands are compared to the Cut Index (specified below). When the vertex index matches the Cut Index any previous topology is terminated. If DISABLED, vertex indices are not compared to the Cut Index. This field can only be enabled for

3DSTATE_VF

		certain primitive topology types. Refer to the table later in this section for details.
	9	**Component Packing Enable** Format: Enable If ENABLED, vertex element component packing (as specified by 3DSTATE_VF_COMPONENT_PACKING) is performed before vertices are written into the URB. If DISABLED, no component packing is performed - all components of valid vertex elements will be stored in the URB.
	8	**Indexed Draw Cut Index Enable** Format: Enable If ENABLED, vertex indices in RANDOM 3DPRIMITIVE commands are compared to the Cut Index (specified below). When the vertex index matches the Cut Index any previous topology is terminated.If DISABLED, vertex indices are not compared to the Cut Index and are used strictly as indices into vertex buffers.This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details.
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Cut Index** This field specifies the index value that is considered the "cut index" which vertex indices are compared to if a Cut Index Enable is set. The Cut Index is compared to the fetched (and possibly-sign-extended) vertex index, and if these values are equal, the current primitive topology is terminated. Note that, for index buffers less than 32bpp, it is possible to set the Cut Index to a (large) value that will never match a sign-extended vertex index.

3DSTATE_VF_INSTANCING

3DSTATE_VF_INSTANCING

Source: RenderCS

Length Bias: 2

This command is used to control the "instancing" state associated with a specific vertex element.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 49h 3DSTATE_VF_INSTANCING Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 1h — Excludes DWord (0,1) **[Default]** Value: 43h — Context Restore
1	31:9	**Reserved** Format: MBZ
	8	**Instancing Enable** Format: Enable Value: 0h — Name: Disabled — Description: This vertex element is not instanced and therefore vertices within instances can each receive different data for this vertex element. Within each instance, the source vertex data for this vertex element is determined according the the Vertex Access Type of the 3DPRIMITIVE command. Instance Data Step Rate is

3DSTATE_VF_INSTANCING

		ignored for this vertex element.
1h	Enabled	This vertex element is instanced and therefore vertices within instances will receive the same data for this vertex element. The source pointer for this particular vertex element will be (a) initialized at the start of 3DPRIMITIVE processing, (b) held constant for all vertices within an instance, and (c) advanced between instances as a function of Instance Data Step Rate.

7:6 Reserved

Format:	MBZ

5:0 Vertex Element Index

Format:	U6

This field identifies which vertex element state is to be modified by this command.

Value	Name
[0,33]	

2 31:0 Instance Data Step Rate

If Instancing Enable is ENABLED, this field determines the rate at which data for this particular vertex element is changed between instances. Only after the number of instances specified by this field is generated is new (sequential) vertex element data provided. This process continues for each group of instances defined in the 3DPRIMTIVE command. For example, a value of 1 in this field causes new data to be supplied for this vertex element with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new vertex element data. The special value of 0 causes all vertices of all instances generated by the 3DPRIMITIVE command to be provided with the same data for this vertex element. (The same effect can be achieved by setting this field to its maximum value.) If Instancing Enable is DISABLED, this field is ignored.

3DSTATE_VF_SGVS

3DSTATE_VF_SGVS

Source: RenderCS

Length Bias: 2

Description

This command is used to control the insertion of the VertexID and InstanceID System-Generated Values (SGVs) into an input Vertex URB Entry (VUE) (available as input to a VS thread). VertexID and InstanceID insertion can be individually controlled. The insertion locations are specified as 128-bit element locations (starting at the beginning of the VUE) and the 32-bit component within those specified elements. The SGV values can be inserted either (a) within a valid vertex element (in which case the value overwrites the value specified via 3DSTATE_VERTEX_ELEMENTS) or (b) beyond the last valid vertex element written to the URB. This permits some orthogonality between the programming of vertex elements (which typically is known at draw time) and programming of SGV insertion (which is associated with the shader). There are some restrictions however (see below). If an SGV is inserted beyond the last valid vertex element, zeroes are first inserted in the VUE after the last valid vertex element up to and including the vertex element receiving an SGV. If both of the SGVs are enabled for insertion, the zeroes will extend to the last (largest index) vertex element receiving an SGV. Then the SGV(s) are inserted.

The insertion of SGV values occurs before any component packing (3DSTATE_VF_COMPONENT_PACKING). Therefore the Element Offsets and Component Numbers specified in this command refer to the pre-packed data, following 3DSTATE_VERTEX_ELEMENT processing.

Programming Notes

Programming Restrictions:

- It is INVALID to store both the VertexID and InstanceID in the same element/component location within the VUE.
- The states programmed by this command overwrite the state programmed by any previous commands. I.e., VertexID and InstanceID (if enabled) can only be inserted in one component of a vertex.
- It is INVALID to insert an SGV value past the end of the VUE entry (as determined by VS URB Entry Allocation Size) or past the 33rd vertex element. Therefore the programming of VS URB Entry Allocation Size needs to comprehend any SGV insertion requirements.
- It is INVALID to use this command to overwrite any portion of a 64-bit vertex element component.
- It is INVALID to use this command to overwrite a EdgeFlag vertex element component or any vertex element beyond it.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D

3DSTATE_VF_SGVS

		Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Ah 3DSTATE_VF_SGVS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**InstanceID Enable** Format: Enable **Value** / **Name** / **Description** 0h / Disabled / InstanceID is not inserted 1h / Enabled / InstanceID is inserted
	30:29	**InstanceID Component Number** If InstanceID Enable is ENABLED, this field specifies the 32-bit component location (within the 4-component VUE) where it is inserted. If InstanceID Enable is DISABLED, this field is ignored. **Value** / **Name** / **Description** 0 / COMP_0 / If enabled, InstanceID is inserted in component 0 (.x) 1 / COMP_1 / If enabled, InstanceID is inserted in component 1 (.y) 2 / COMP_2 / If enabled, InstanceID is inserted in component 2 (.z) 3 / COMP_3 / If enabled, InstanceID is inserted in component 3 (.w)
	28:22	**Reserved** Format: MBZ
	21:16	**InstanceID Element Offset** Format: U6 Offset of 128-bit element If InstanceID Enable is ENABLED, this field specifies the VUE element offset of the 128-bit element where it is to be inserted. The InstanceID Component Number specifies where in the specified element it is inserted. **Value** / **Name** [0,33] /

3DSTATE_VF_SGVS

	15	**VertexID Enable** Format: Enable **Value** / **Name** / **Description** 0h / Disabled / VertexID is not inserted 1h / Enabled / VertexID is inserted
	14:13	**VertexID Component Number** If VertexID Enable is ENABLED, this field specifies the 32-bit component location (within the 4-component VUE) where it is inserted.If VertexID Enable is DISABLED, this field is ignored. **Value** / **Name** / **Description** 0 / COMP_0 / If enabled, VertexID is inserted in component 0 (.x) 1 / COMP_1 / If enabled, VertexID is inserted in component 1 (.y) 2 / COMP_2 / If enabled, VertexID is inserted in component 2 (.z) 3 / COMP_3 / If enabled, VertexID is inserted in component 3 (.w)
	12:6	**Reserved** Format: MBZ
	5:0	**VertexID Element Offset** Format: U6 Offset of 128-bit element If VertexID Enable is ENABLED, this field specifies the VUE element offset of the 128-bit element where it is to be inserted. The VertexID Component Number specifies where in the specified element it is inserted. This is also the vertex element index. If VertexID Enable is DISABLED, this field is ignored. **Value** / **Name** [0,33] /

3DSTATE_VF_STATISTICS

3DSTATE_VF_STATISTICS

Source: RenderCS

Length Bias: 1

The VF stage tracks two pipeline statistics, the number of vertices fetched and the number of objects generated. VF will increment the appropriate counter for each when statistics gathering is enabled by issuing the 3DSTATE_VF_STATISTICS command with the [Statistics Enable] bit set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Format: Opcode Value 1h: GFXPIPE_SINGLE_DW **[Default]**
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	23:16	**3D Command Sub Opcode** Default Value: 0Bh 3DSTATE_VF_STATISTICS Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	15:1	**Reserved** Format: MBZ
	0	**Statistics Enable** Format: Enable If ENABLED, VF will increment the pipeline statistics counters IA_VERTICES_COUNT and IA_PRIMITIVES_COUNT for each vertex fetched and each object output, respectively, for 3DPRIMITIVE commands issued subsequently. If DISABLED, these counters will not be incremented for subsequent 3DPRIMITIVE commands.

3DSTATE_VF_TOPOLOGY

3DSTATE_VF_TOPOLOGY

Source: RenderCS

Length Bias: 2

This command specifies the VF stage's Topology state which can be used to override the Primitive Topology Type in subsequent 3DPRIMITIVE commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Bh 3DSTATE_VF_TOPOLOGY Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:6	**Reserved** Format: MBZ
	5:0	**Primitive Topology Type** Format: 3D_Prim_Topo_Type This field specifies the VF stage's Topology state.

3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_CC

Source: RenderCS

Length Bias: 2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CC command is used to define the location of fixed functions' viewport state table.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 23h 3DSTATE_VIEWPORT_STATE_POINTERS_CC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:5	**CC Viewport Pointer** Format: DynamicStateOffset[31:5]CC_VIEWPORT*16 Specifies the 32-byte aligned address offset of the CC_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	4:0	**Reserved** Format: MBZ

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

Source: RenderCS

Length Bias: 2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CLIP command is used to define the location of fixed functions' viewport state table.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 21h 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:6	**SF Clip Viewport Pointer** Format: DynamicStateOffset[31:6]SF_CLIP_VIEWPORT*16 Specifies the 64-byte aligned address offset of the SF_CLIP_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	5:0	**Reserved** Format: MBZ

3DSTATE_VS

3DSTATE_VS

Source: RenderCS, PositionCS

Length Bias: 2

Description
This command specifies most of the state used by the Vertex Shader (VS) stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 10h 3DSTATE_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 7h Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:6	**Kernel Start Pointer** Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location of the kernel program run by threads spawned by the VS pipeline stage. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if VS Function Enable is DISABLED.
	5:0	**Reserved** Format: MBZ
3	31	**Single Vertex Dispatch** Format: U1 Enumerated Type When this bit is set, SIMD4x2 VS threads will only process a single vertex, otherwise SIMD4x2 threads will process either one or two vertices. This field is ignored if **SIMD8 Dispatch Enable** is

3DSTATE_VS

set.

Value	Name	Description
0h	Multiple	Dual vertex SIMD4x2 thread dispatches are allowed.
1h	Single	Single vertex SIMD4x2 thread dispatches are forced.

30 **Vector Mask Enable**

Format:	Enable

Upon subsequent VS thread dispatches, this bit is loaded into the EU's **Vector Mask Enable** (VME, cr0.0[3]) thread state. Refer to EU documentation for the definition and use of VME state.

Value	Name	Description
0h	Dmask	The EU will use the Dispatch Mask (supplied by the VS stage) for instruction execution.
1h	Vmask	The EU will use the Vector Mask (derived from the Dispatch Mask) for instruction execution.

Programming Notes
Under normal conditions SW shall specify DMask, as the VS stage will provide a Dispatch Mask appropriate to SIMD4x2 or SIMD8 thread execution (as a function of SIMD8 Dispatch Enable). E.g., for SIMD4x2 thread execution, the VS stage will generate a Dispatch Mask that is equal to what the EU would use as the Vector Mask. For SIMD8 execution there is no known usage model for use of Vector Mask (as there is for PS shaders).

29:27 **Sampler Count**

Format:	U3

This field specifies (in multiples of 4) the number of sets of sampler state that will be prefetched for use by the VS kernel. While the prefetching of sampler state is optional and does not impact functionality, it may improve performance.
This field is ignored if the Function Enable state is set to DISABLED.

Value	Name	Description
0h	No Samplers	no samplers used
1h	1-4 Samplers	between 1 and 4 samplers used
2h	5-8 Samplers	between 5 and 8 samplers used
3h	9-12 Samplers	between 9 and 12 samplers used
4h	13-16 Samplers	between 13 and 16 samplers used

26 **Reserved**

Format:	MBZ

25:18 **Binding Table Entry Count**

Format:	U8

Description
Specifies how many binding table entries the kernel uses. Used only for prefetching of the

3DSTATE_VS

binding table entries and associated surface state.
Note: For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.
This field is ignored if VS Function Enable is DISABLED.

When HW Generated Binding Table bit is enabled: This field indicates which cache lines (512bit units - 32 Binding Table Entry section) should be fetched. Each bit in this field corresponds to a cache line. Only the 1st 4 non-zero Binding Table entries of each 32 Binding Table entry section prefetched will have its surface state prefetched.

Value	Name
[0,255]	

Programming Notes
When HW binding table bit is set, it is assumed that the Binding Table Entry Count field will be generated at JIT time.

17 **Thread Dispatch Priority**

Format:	U1 Enumerated Type

Specifies the priority of the thread for dispatch:
This field is ignored if VS Function Enable is DISABLED.

Value	Name	Description
0h	Normal	Normal Priority
1h	High	High Priority

16 **Floating Point Mode**

Format:	U1 Enumerated Type

Specifies the initial floating point mode used by the dispatched thread.
This field is ignored if VS Function Enable is DISABLED.

Value	Name	Description
0h	IEEE-754	Use IEEE-754 Rules
1h	Alternate	Use Alternate Rules

15:14 **Reserved**

Format:	MBZ

13 **Illegal Opcode Exception Enable**

Format:	Enable

This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.
This field is ignored if VS Function Enable is DISABLED.

3DSTATE_VS

	12	**Accesses UAV** Format: Enable This field must be set when VS has a UAV access. **Programming Notes** This field must not be set when VS Function Enable is disabled.
	11:8	**Reserved** Format: MBZ
	7	**Software Exception Enable** Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if VS Function Enable is DISABLED.
	6:0	**Reserved** Format: MBZ
4..5	63:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[63:10]ScratchSpace Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if VS Function Enable is DISABLED. In 64b OS all pointers need to be seen by SW as 48b. HW does not support a Scratch Space Base Pointer larger than 32b, therefore SW must ensure Bits<63:32> are set to 0's.
	9:4	**Reserved** Format: MBZ
	3:0	**Per-Thread Scratch Space** Format: U4 power of 2 Bytes over 1K Bytes Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if VS Function Enable is DISABLED. **Value** / **Name** / **Description** [0,11] / / Indicating [1K Bytes, 2M Bytes] **Programming Notes**

3DSTATE_VS

		This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.
6	31:25	**Reserved** Format: MBZ
	24:20	**Dispatch GRF Start Register For URB Data** Format: U5 Specifies the starting GRF number for the URB portion (URB constants and vertices) of the thread payload. This field is ignored if VS Function Enable is DISABLED. **Value** / **Name** / **Description** [0,31] / / indicating GRF [R0, R31]
	19:17	**Reserved** Format: MBZ
	16:11	**Vertex URB Entry Read Length** Format: U6 Specifies the number of pairs of 128-bit vertex elements to be passed into the payload for each vertex. This field is ignored if VS Function Enable is DISABLED. For SIMD4x2 dispatch, each vertex element requires one GRF of payload data, therefore the number of GRFs with vertex data will be double the value programmed in this field. For SIMD8 dispatch, each vertex element requires 4 GRFs of payload data, therefore the number of GRFs with vertex data will be 8 times the value programmed in this field. The EU limit of 128 GRFs imposes a maximum limit of 30 elements per vertex pushed into the payload, though the practical limit may be lower. If input vertices exceed the practical limit, software must decide between resorting to pulling elements during thread execution or dropping back to SIMD4x2 dispatch. Note that the VUE is used for both input and output, so when using the pull-model software must ensure inputs are not overwritten before last use. **Value** / **Name** / **Description** [1,63] / / if SIMD8 dispatch disabled [0,15] / / if SIMD8 dispatch enabled
	10	**Reserved** Format: MBZ
	9:4	**Vertex URB Entry Read Offset** Format: U6 Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if VS Function Enable is DISABLED. **Value** / **Name**

3DSTATE_VS

		[0,63]
	3:0	**Reserved** Format: MBZ
7	31:23	**Maximum Number of Threads** Format: U9-1 Thread count Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance. This field is ignored if VS Function Enable is DISABLED. **Value** / **Name** / **Description** [0,335] / / indicating thread count of [1,336]
	22	**Reserved** Format: MBZ
	21:13	**Reserved** Format: MBZ
	12:11	**Reserved** Format: MBZ
	10	**Statistics Enable** Format: Enable If ENABLED, the VS stage will perform statistics gathering. See the Statistics Gathering subsection. If DISABLED, statistics information associated with the VS stage will be left unchanged.
	9	**Reserved** Format: MBZ
	8:3	**Reserved** Format: MBZ
	2	**SIMD8 Dispatch Enable** Format: Enable This field determines how VS threads are dispatched and how the thread payloads are generated. The setting of this field must agree with how the VS kernel was compiled. If ENABLED, SIMD8 VS thread dispatches are performed. The **Single Vertex Dispatch** field is ignored. If DISABLED, SIMD4x2 thread dispatches are performed. The **Single Vertex Dispatch** field can be used to force single-vertex dispatches.

3DSTATE_VS

	1	**Vertex Cache Disable** Format: Disable This bit controls the operation of the Vertex Cache. This field is always used. If the Vertex Cache is DISABLED and the VS Function is ENABLED, the Vertex Cache is not used and all incoming vertices will be passed to VS threads. If the Vertex Cache is ENABLED and the VS Function is ENABLED, only incoming vertices that do not hit in the Vertex Cache will be passed to VS threads. If the Vertex Cache is ENABLED and the VS Function is DISABLED, input vertices that miss in the Vertex Cache will be assembled and written to the URB (by the VF stage), and subsequently passed through the VS stage unmodified (i.e, no VS threads are spawned). The Vertex Cache is invalidated whenever the Vertex Cache becomes DISABLED, whenever the VS Function Enable toggles, between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command.
	0	**Function Enable** Format: Enable This bit determines whether or not the VS stage spawns VS threads, which comprises the bulk of the VS stage functionality. If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline. If DISABLED, VF-generated vertices will pass thru the VS function and are sent down the pipeline unmodified. The Vertex Cache (if enabled) is still available.
8	31:28	**Reserved** Format: MBZ
	27	**Reserved** Format: MBZ
	26:21	**Vertex URB Entry Output Read Offset** Format: U6 Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB by the Setup Back-End (SBE) function. The offset programmed will specify the start of Attribute 0 to be passed in subsequent Pixel Shader thread payloads. Refer to the Attribute Interpolator Setup documentation.

Value	Name
[0,63]	

Programming Notes

As the vertex header data located at the start of the Vertex URB entry is typically only used by 3D pipeline FFs (i.e., Clipper, Setup FrontEnd) and not required as interpolated attributes in PIxel Shader threads, it is expected that SW will program this Start Offset skip over the vertex header.

This offset value is ignored if SBE's Number of SF Attributes state is programmed to 0 (i.e., no

3DSTATE_VS

		attributes are defined beyond the position read from the Vertex Header)
	20:16	**Vertex URB Entry Output Length** Format: U5 Specifies the amount of Vertex Attribute URB data to be read by the Setup Back-End function for each Vertex URB entry, in 256-bit units. The attribute data will be read starting at the offset specified by the Vertex URB Entry Output Read Offset state. **Value** [1,16] — **Name** **Programming Notes** This length value is ignored if SBE's Number of SF Attributes state is programmed to 0 (i.e., no attributes are defined beyond the position read from the Vertex Header).
	15:8	**User Clip Distance Clip Test Enable Bitmask** Format: U8 This 8 bit mask field selects which of the 8 Clip Distance Values (if any) are to be included in the Clip stage's trivial reject / trivial accept / must clip determination function. The ClipDistance Values (if present) are located in DW8-15 of the VUE Vertex Header located at the beginning of VUE URB entries. Bit 0 of this field corresponds to Clip Distance Value 0.
	7:0	**User Clip Distance Cull Test Enable Bitmask** Format: U8 This 8 bit mask field selects which of the 8 Clip Distance Values (if any) are to be included in the Clip stage's trivial reject / trivial accept determination function. Note that must clip determination is not included in this function. The ClipDistance Values (if present) are located in DW8-15 of the VUE Vertex Header located at the beginning of VUE URB entries. Bit 0 of this field corresponds to Clip Distance Value 0.

3DSTATE_WM_CHROMAKEY

3DSTATE_WM_CHROMAKEY

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Ch 3DSTATE_WM__CHROMAKEY Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31	**ChromaKey Kill Enable** Format: Enable If ENABLED, indicates that at least one of the attached samplers has ChromaKeyKill enabled.
	30:0	**Reserved** Format: MBZ

3DSTATE_WM_DEPTH_STENCIL

3DSTATE_WM_DEPTH_STENCIL

Source: RenderCS

Length Bias: 2

This command replaces the indirect state DEPTH_STENCIL_STATE with an inline state command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Eh 3DSTATE_WM_DEPTH_STENCIL Format: OpCode
	15:13	**Reserved** Format: MBZ
	12:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Format: =n Total Length - 2 Value: 02h — Name: Excludes Dword (0,1) **[Default]**
1	31:29	**Stencil Fail Op** Format: 3D_Stencil_Operation This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test fails. **Programming Notes** if all three stencil ops (Stencil Fail, Stencil Pass Depth Fail, and Stencil Pass Depth Pass) are KEEP, ZERO, or REPLACE, the stencil buffer is not read.
	28:26	**Stencil Pass Depth Fail Op**

3DSTATE_WM_DEPTH_STENCIL

		Format: 3D_Stencil_Operation This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth pass fails.
	25:23	**Stencil Pass Depth Pass Op** Format: 3D_Stencil_Operation This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth test passes.
	22:20	**Backface Stencil Test Function** Format: 3D_Compare_Function
	19:17	**Backface Stencil Fail Op** Format: 3D_Stencil_Operation
	16:14	**Backface Stencil Pass Depth Fail Op** Format: 3D_Stencil_Operation This field specifies the operation to perform on the Stencil Buffer when the stencil test passes but the depth pass fails.
	13:11	**Backface Stencil Pass Depth Pass Op** Format: 3D_Stencil_Operation This field specifies the operation to perform on the Stencil Buffer when the stencil test passes and the depth pass passes (or is disabled).
	10:8	**Stencil Test Function** Format: 3D_Compare_Function This field specifies the comparison function used in the (front face) StencilTest function.
	7:5	**Depth Test Function** Format: 3D_Compare_Function Specifies the comparison function used in DepthTest function. **Programming Notes** If the Depth Test Function is ALWAYS or NEVER, the depth buffer is not read.
	4	**Double Sided Stencil Enable** Format: Enable Enable doubled sided stencil operations.

Value	Name	Description
0h	False	Double Sided Stencil Disabled
1h	True	Double Sided Stencil Enabled

3DSTATE_WM_DEPTH_STENCIL

		Programming Notes • Back-facing primitives have a vertex winding order opposite to the currently selected Front Winding state. • Culling of primitives is not affected by the double sided stencil state • Back-facing primitives will be rendered, honoring all current device state, as though it were a front-facing primitive with no implicitly overloaded state.
	3	**Stencil Test Enable** Format: Enable Enables StencilTest function of the Pixel Processing pipeline. **Programming Notes** If any of the render targets are YUV format, this field must be disabled.
	2	**Stencil Buffer Write Enable** Format: Enable Enables writes to the Stencil Buffer. **Programming Notes** If this field is enabled, Stencil Test Enable must also be enabled.
	1	**Depth Test Enable** Format: Enable Enables the DepthTest function of the Pixel Processing pipeline. **Value** / **Name** 0h – Disable 1h – Enable **Programming Notes** If any of the render targets are YUV format, this field must be disabled.
	0	**Depth Buffer Write Enable** Format: Enable Enables writes to the Depth Buffer. **Programming Notes** A Depth Buffer must be defined before enabling writes to it, or operation is UNDEFINED. This bit must not be set when WM_INT::RT Independent Rasterization Enable is true.
2	31:24	**Stencil Test Mask** Format: U8 This field specifies a bit mask applied to stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.

3DSTATE_WM_DEPTH_STENCIL

	23:16	**Stencil Write Mask** Format: U8 This field specifies a bit mask applied to stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.
	15:8	**Backface Stencil Test Mask** Format: U8 This field specifies a bit mask applied to backface stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.
	7:0	**Backface Stencil Write Mask** Format: U8 This field specifies a bit mask applied to backface stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.
3	31:16	**Reserved** Format: MBZ
	15:8	**Stencil Reference Value** Format: U8 This field specifies the stencil reference value to compare against in the (front face) StencilTest function.
	7:0	**Backface Stencil Reference Value** Format: U8 This field specifies the stencil reference value to compare against in the StencilTest function.

3DSTATE_WM

3DSTATE_WM

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_WM Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**Statistics Enable** Format: Enable If ENABLED, the Windower and pixel pipeline will engage in statistics gathering. If DISABLED, statistics information associated with this FF stage will be left unchanged. See Statistics Gathering. **Programming Notes** This bit must be disabled if any of these bits is set: 3DSTATE_WM::**Legacy Depth Buffer Clear**, 3DSTATE_WM::**Legacy Hierarchical Depth Buffer Resolve Enable** or 3DSTATE_WM::**Legacy Depth Buffer Resolve Enable**.
	30	**Legacy Depth Buffer Clear Enable** Format: Enable When set, the depth buffer is initialized as a side-effect of rendering pixels.

3DSTATE_WM

Programming Notes

If this field is enabled,

1. the **Depth Test Enable** field in DEPTH_STENCIL_STATE must be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.
3. 3DSTATE_DEPTH_BUFFER::Stencil Write Enable must be set if 3DSTATE_STENCIL_BUFFER::Stencil buffer enable is set. Additionally the following must be set to the correct values.
    1. DEPTH_STENCIL_STATE::Stencil Write Mask must be 0xFF
    2. DEPTH_STENCIL_STATE::Stencil Test Mask must be 0xFF
    3. DEPTH_STENCIL_STATE::Back Face Stencil Write Mask must be 0xFF
    4. DEPTH_STENCIL_STATE::Back Face Stencil Test Mask must be 0xFF

Refer to section 0 "Depth Buffer Clear" for additional restrictions when this field is enabled. If this field is enabled,**Pixel Shader Kill Pixel** must be disabled.

29 **Reserved**

Format:	MBZ

28 **Legacy Depth Buffer Resolve Enable**

Format:	Enable

When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Legacy Depth Buffer Clear** and **Legacy Hierarchical Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

Refer to section 11.5.4.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled.
If Hierarchical Depth Buffer Enable is disabled, enabling this field will have no effect.

27 **Legacy Hierarchical Depth Buffer Resolve Enable**

Format:	Enable

When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Legacy Depth Buffer Clear** and **Legacy Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

3DSTATE_WM

Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled.

If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect.

Performance Note: expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.

26 **Legacy Diamond Line Rasterization**

Format:	Enable

This bit, if ENABLED, indicates that the Windower will rasterize zero width lines using the DX9 rasterization rules. If DISABLED, the Windower will rasterize zero width lines using the DX10 rasterization rules (see Strips Fans chapter).

25:23 **Reserved**

Format:	MBZ

22:21 **Early Depth/Stencil Control**

Format:	U2 Enumerated Type

This field specifies the behavior of early depth/stencil test.

Value	Name	Description
0h	NORMAL	Depth/Stencil Test/Write behaves as if it happens post-shader, however the pixel shader is not necessarily executed if the pixel fails depth or stencil test (this is the legacy behavior)
1h	PSEXEC	Depth/Stencil Test/Write behaves as if it happens post-shader, and the pixel shader is executed if the pixel fails depth or stencil test (although pre-shader actions such as primitive inclusion, stipple, etc. will still cause the shader not to execute)
2h	PREPS	Depth/Stencil Test/Write behaves as if it happens pre-shader. The pixel shader is not executed if the pixel fails depth or stencil test. Depth and stencil writes occur even if the pixel is killed by the shader or post-shader by alpha test, etc. Depth output by the pixel shader is ignored.
3h	Reserved	

Programming Notes

The Early Depth/Stencil Control field cannot be set to PREPS (value = 2h) if ForceKillpix = ForceON or Forced Thread Dispatch = ForceON

20:19 **Force Thread Dispatch Enable**

Value	Name	Description
0h	Normal	WM_INT::ThreadDispatchEnable is computed normally
1h	ForceOff	Forces WM_INT::ThreadDispatchEnable Off
2h	ForceON	Forces WM_INT::ThreadDispatchEnable On

3DSTATE_WM

3h	Reserved	

Programming Notes

This should must always be set to Normal. This field should not be tested for functional validation

18:17 **Position ZW Interpolation Mode**

Format:	U2 Enumerated Type

This field elects "interpolation mode" associated with the Position Z (source depth) and W coordinates passed in the PS payload when the PS requires Position as input. This field does not determine whether these coordinates are actually included in the payload (see Pixel Shader Requires Depth, Pixel Shader Requires W).

Value	Name	Description
0h	INTERP_PIXEL	Evaluate Z & W at the pixel center or UL corner (as specified by Pixel Location of 3DSTATE_MULTISAMPLE)
1h	Reserved	
2h	INTERP_CENTROID	
3h	INTERP_SAMPLE	

Programming Notes

WM_INT::RT Independent Rasterization Enable must be disabled in order to select INTERP_SAMPLE.

MSDISPMODE_PERSAMPLE is required in order to select INTERP_SAMPLE.

16:11 **Barycentric Interpolation Mode**

Format:	Enable[6]

Controls which barycentric interpolation terms must be passed into the pixel shader kernel.
Bit 0: Perspective Pixel Location barycentric is required
Bit 1: Perspective Centroid barycentric is required
Bit 2: Perspective Sample barycentric is required
Bit 3: Non-perspective Pixel Location barycentric is required
Bit 4: Non-perspective Centroid barycentric is required
Bit 5: Non-perspective Sample barycentric is required

Programming Notes

If contiguous dispatch modes are enabled, only bit 3 (non-perspective pixel location) can be set, all other bits in this field must be zero.

Pixel Location below refers to either the upper left corner or pixel center depending on the **Pixel Location** state of 3DSTATE_MULTISAMPLING).

MSDISPMODE_PERSAMPLE is required in order to select Perspective Sample or Non-perspective Sample barycentric coordinates.

10 **Reserved**

Format:	MBZ

3DSTATE_WM

9:8 Line End Cap Antialiasing Region Width

Format:	U2

This field specifies the distances over which the coverage of anti-aliased line end caps are computed.

Value	Name	Description
0h	0.5 pixels	0.5 pixels
1h	1.0 pixels	1.0 pixels
2h	2.0 pixels	2.0 pixels
3h	4.0 pixels	4.0 pixels

7:6 Line Antialiasing Region Width

Format:	U2

This field specifies the distance over which the anti-aliased line coverage is computed.

Value	Name	Description
0h	0.5 pixels	0.5 pixels
1h	1.0 pixels	1.0 pixels
2h	2.0 pixels	2.0 pixels
3h	4.0 pixels	4.0 pixels

5 Reserved

Format:	MBZ

4 Polygon Stipple Enable

Format:	Enable

Enables the Polygon Stipple function.

3 Line Stipple Enable

Format:	Enable

Enables the Line Stipple function.

2 Point Rasterization Rule

Format:	3D_RasterizationRule

This field specifies the rasterization rules to be applied whenever the edges of a point primitive fall exactly on a pixel sampling point.

Value	Name	Description
0h	RASTRULE_UPPER_LEFT	To match "normal" upper left rules for surface primitives
1h	RASTRULE_UPPER_RIGHT	To match OpenGL point rasterization rules (round to + infinity, where this is the upper right direction wrt OpenGL screen origin of lower left).

1:0 Force Kill Pixel Enable

Value	Name	Description

3DSTATE_WM

0h	Normal	WM_INT:: Pixel Shader Kill Pixel is computed normally
1h	ForceOff	Forces WM_INT:: Pixel Shader Kill Pixel Off
2h	ForceON	Forces WM_INT:: Pixel Shader Kill Pixel On
3h	Reserved	

Programming Notes

This should must always be set to Normal. This field should not be tested for functional validation

3DSTATE_WM_HZ_OP

3DSTATE_WM_HZ_OP

Source: RenderCS

Length Bias: 2

This command provides for clearing Z and/or stencil or resolving either HZ buffer or Z buffer.

Programming Notes

As this command generates an implicit rectangle, SW must make sure any MMIO register writes following WM_HZ_OP must be preceded by **PIPE_CONTROL** with **Command Streamer Stall Enable** bit set.

3DSTATE_MULTISAMPLE packet must be used prior to this packet to change the Number of Multisamples. This packet must not be used to change Number of Multisamples in a rendering sequence.
3DSTATE_RASTER if used must be programmed prior to using this packet.

This command does NOT support predication from the use of the MI_PREDICATE register. To predicate depth clears and resolves on you must fall back to using the 3D_PRIMITIVE or GPGPU_WALKER commands.

Clear_Value_Test_Enable	Force_Slow_Clear_Enable	Clear_Value_Write_Enable	
0	0	0	Fast Clear; No Clear_value_address update
0	0	1	Fast Clear; Clear_value_address updated
0	1	0	Slow Clear; No Clear_value_address update
0	1	1	Illegal
1	0	0	Conditional Fast Clear; No Clear_value_address update
1	1	0	Illegal
1	X	1	Illegal

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_WM_HZ_OP

	23:16	**3D Command Sub Opcode** Default Value: 52h 3DSTATE_WM_HZ_OP Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Format: =n Total Length - 2 Value: 03h — Name: Excludes Dword (0,1) **[Default]**
1	31	**Stencil Buffer Clear Enable** Format: Enable When set, the stencil buffer is initialized. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Resolve Enable** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Stencil Write Enable must be set if 3DSTATE_STENCIL_BUFFER::Stencil buffer enable is set.
	30	**Depth Buffer Clear Enable** Format: Enable When set, the depth buffer is initialized. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Resolve Enable** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.
	29	**Scissor Rectangle Enable** Format: Enable Enables operation of Scissor Rectangle. **Programming Notes** In order get the functionality right if this bit is disabled, driver must clip the clear rectangle to scissor rectangle if scissor test is

3DSTATE_WM_HZ_OP

		enabled before clearing. **Workaround** Workaround: Due to a Hardware issue this bit must not be set.
	28	**Depth Buffer Resolve Enable** Format: Enable When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Clear** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set. Refer to section 11.5.4.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled. If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect.
	27	**Hierarchical Depth Buffer Resolve Enable** Format: Enable When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified outside of the 3D rendering operation. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Clear** and **Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set. Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled. If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect. **Performance Note:** expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more

3DSTATE_WM_HZ_OP

		effective state.
	26	**Pixel Position Offset Enable** Format: Enable Enumerated Type Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions. **Programming Notes** Setting this field along with setting the Pixel Location to upper left and number of multisamples to greater than one will cause the device to offset pixel postions by 0.5 both in horizontal and vertical directions.It is to be noted this is done to adjust the pixel co-ordinate system to DX9 like, so any WM_HZ_OP screen space rectangles (eg: legacy HiZ Clear, Resolve etc) generated internally by driver in this mode needs to be aware of this offset adjustment and send the rectangles according to alignment restriction taking this offset adjustment into consideration. SW can choose to set this bit only for DX9 API. DX10/OGL API's should not have any effect by setting or not setting this bit.
	25	**Full Surface Depth and Stencil Clear** Format: Enable **Programming Notes** Setting this field to "1" along with "Depth buffer clear" will cause all the pixels/samples in an the HZ and Stencil CLs to be cleared. Software must set this only when the APP requires the entire Depth surface to be cleared. Setting this field to "1" for STC-buffer only clear without "depth buffer clear" will cause all the pixels/samples in the STC-CL to get the stc-ref value.
	24	**Reserved** Format: MBZ
	23:16	**Stencil Clear Value** Format: U8.0 This field specifies the stencil clear value.
	15:13	**Number of Multisamples** Format: U3 Enumerated Type This field specifies how many samples/pixel exist in the Depth Buffer and Stencil buffers, as log2(#samples).

Value	Name	Description
0h	1	1 sample/pixel
1h	2	2 samples/pixel

3DSTATE_WM_HZ_OP

		2h: 4: 4 samples/pixel 3h: 8: 8 samples/pixel 4h: 16: 16 samples/pixel 5h-7h: Reserved
	12:0	**Reserved**
2 **Programming Notes:** The clear rectangle x and y min and max values must be shifted by the LOD level; i.e. the hardware does not include the LOD in this function. Hence to clear any particular X, Y from the base level, to clear the contents at level "LOD" use (X»LOD) and (Y»LOD).	31:16	**Clear Rectangle Y Min** Format: U16 in Pixels from Depth Buffer origin (upper left corner) Specifies Ymin value of (inclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with Y coordinates less than Ymin will not be affected. **Value** / **Name** [0,16383] / (Device ignores bits 31:30)
	15:0	**Clear Rectangle X Min** Format: U16 in Pixels from Depth Buffer origin (upper left corner) Specifies Xmin value of (inclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with X coordinates less than or equal to Xmin will not be affected. **Value** / **Name** [0,16383] / (Device ignores bits 15:14)
3 **Programming Notes:** See the programming note in the previous DWORD for the Min values. The clear rectangle x and y min and max values must be shifted by the LOD level; i.e. the hardware does not include the LOD in this function. Hence to clear any particular X, Y from the base level, to clear the contents at level "LOD" use (X»LOD) and (Y»LOD). Hence the max values must be less than or equal to: (Surface Width » LOD) and (Surface Height » LOD) for X Max and Y Max respectively.	31:16	**Clear Rectangle Y Max** Format: U16 in Pixels from Depth Buffer origin (lower right corner) Specifies Ymax value of (exclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with Y coordinates greater than Ymax will be not be cleared. **Value** / **Name** [0,16383] / (Device ignores bits 31:30)
	15:0	**Clear Rectangle X Max** Format: U16 in Pixels from Depth Buffer origin (lower right corner) Specifies Xmax value of (exclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with X coordinates greater than or equal to Xmax will be not be affected. **Value** / **Name** [0,16383] / (Device ignores bits 15:14)

3DSTATE_WM_HZ_OP		
4	31:16	**Reserved** Format: MBZ
	15:0	**Sample Mask** Format: Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_WM_HZ_OP) A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection. **Programming Notes** If Number of Multisamples is NUMSAMPLES_1, bits 15:1 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_2, bits 15:2 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_4, bits 15:4 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_8, bits 15:8 of this field will be zeroed by HW.

A64 Byte Scaled Read MSD

MSD2R_A64_BS - A64 Byte Scaled Read MSD

Source:	DataPort 2
Length Bias:	1
Family:	Scaled R/W
Group:	Byte Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 02h Format: Opcode A64 Scattered Read message
	14	**Reserved** Format: MBZ Ignored
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype

MSD2R_A64_BS - A64 Byte Scaled Read MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Byte Scaled Write MSD

MSD2W_A64_BS - A64 Byte Scaled Write MSD

Source: DataPort 2
Length Bias: 1
Family: Scaled R/W
Group: Byte Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 0Bh Format: Opcode A64 Scattered Write message
	14	**Reserved** Format: MBZ Ignored
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype

MSD2W_A64_BS - A64 Byte Scaled Write MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Byte Scattered Read MSD

MSD1R_A64_BS - A64 Byte Scattered Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Scattered R/W
Group:	Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Byte Scattered Write MSD

MSD1W_A64_BS - A64 Byte Scattered Write MSD

Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Scaled Read MSD

MSD2R_A64_DWS - A64 Dword Scaled Read MSD

Source:	DataPort 2
Length Bias:	1
Family:	Scaled R/W
Group:	DW Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 02h Format: Opcode A64 Scattered Read message
	14	**Reserved** Format: MBZ Ignored
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype

MSD2R_A64_DWS - A64 Dword Scaled Read MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Dword Scaled Write MSD

MSD2W_A64_DWS - A64 Dword Scaled Write MSD

Source:	DataPort 2
Length Bias:	1
Family:	Scaled R/W
Group:	DW Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 0Bh Format: Opcode A64 Scattered Write message
	14	**Reserved** Format: MBZ Ignored
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype

MSD2W_A64_DWS - A64 Dword Scaled Write MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Dword Scattered Read MSD

MSD1R_A64_DWS - A64 Dword Scattered Read MSD

Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Scattered Write MSD

MSD1W_A64_DWS - A64 Dword Scattered Write MSD

Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD

MSD1R_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD

MSD1W_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD

	9:8	**Atomic Float Operation Type** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD

MSD1R_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD		
	9:8	**Atomic Float Operation** Format: **MDC_FOP3** Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

MSD1W_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float Binary with Return Data Operation MSD

MSD1R_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2S Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary with Return Data Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float Binary Write Only Operation MSD

MSD1W_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2S Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary Write Only Operation MSD

	9:8	**Atomic Float Operation Type** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float Trinary with Return Data Operation MSD

MSD1R_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2S Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary with Return Data Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float Trinary Write Only Operation MSD

MSD1W_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2S Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary Write Only Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Hword Block Read MSD

MSD1R_A64_HWB - A64 Hword Block Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 3h Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_HW Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Hword Block Write MSD

MSD1W_A64_HWB - A64 Hword Block Write MSD

Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Hword Block Write message
	13	**Reserved** Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 3h Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_HW Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Block Read MSD

MSD1R_A64_OWB - A64 Oword Block Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 0h Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Block Write MSD

MSD1W_A64_OWB - A64 Oword Block Write MSD

Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Oword Block Write message
	13	**Reserved** Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 0h Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Dual Block Read MSD

MSD1R_A64_OWDB - A64 Oword Dual Block Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 2h Format: Opcode Oword Dual Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Dual Block Write MSD

MSD1W_A64_OWDB - A64 Oword Dual Block Write MSD

Source: DataPort 1
Length Bias: 1
Family: Block R/W
Group: OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Oword Block Write message
	13	**Reserved** Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 2h Format: Opcode Oword Dual Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Unaligned Block Read MSD

MSD1R_A64_OWUB - A64 Oword Unaligned Block Read MSD

Source: DataPort 1
Length Bias: 1
Family: Block R/W
Group: OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Reserved** Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 1h Format: Opcode Oword Unaligned Block Read subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Scaled Read MSD

MSD2R_A64_QWS - A64 Qword Scaled Read MSD

Source: DataPort 2
Length Bias: 1
Family: Scaled R/W
Group: QW Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 02h Format: Opcode A64 Scattered Read message
	14	**Reserved** Format: MBZ Ignored
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype

MSD2R_A64_QWS - A64 Qword Scaled Read MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Qword Scaled Write MSD

MSD2W_A64_QWS - A64 Qword Scaled Write MSD

Source:	DataPort 2
Length Bias:	1
Family:	Scaled R/W
Group:	QW Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 0Bh Format: Opcode A64 Scattered Write message
	14	**Reserved** Format: MBZ Ignored
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype

MSD2W_A64_QWS - A64 Qword Scaled Write MSD		
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Qword Scattered Read MSD

MSD1R_A64_QWS - A64 Qword Scattered Read MSD

Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: QW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Scattered Write MSD

MSD1W_A64_QWS - A64 Qword Scattered Write MSD

Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: QW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Format: MBZ Ignored
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Specifies the atomic integer operation to be performed. Workaround Workaround : CMPWR_2W is not supported in A64 SIMD4x2
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP3S Specifies the atomic integer operation to be performed. **Workaround** Workaround : CMPWR_2W is not supported in A64 QWord SIMD8.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Scaled Untyped Surface Read MSD

MSD2R_A64_US - A64 Scaled Untyped Surface Read MSD

Source:	DataPort 2
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scaled Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 03h Format: Opcode A64 Untyped Surface Read message
	14	**Reserved** Format: MBZ Ignored
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Scaled Untyped Surface Write MSD

MSD2W_A64_US - A64 Scaled Untyped Surface Write MSD

Source:	DataPort 2
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scaled Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A64_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 0Ah Format: Opcode A64 Untyped Surface Write message
	14	**Reserved** Format: MBZ Ignored
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Sideband Scaled Offset** Format: MDC_A64_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

A64 Untyped Surface Read MSD

MSD1R_A64_US - A64 Untyped Surface Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 11h Format: Opcode A64 Untyped Surface Read message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Untyped Surface Write MSD

MSD1W_A64_US - A64 Untyped Surface Write MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 19h Format: Opcode A64 Untyped Surface Write message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

Addition

add - Addition

Source: Eulsa

Length Bias: 4

The add instruction performs component-wise addition of src0 and src1 and stores the results in dst. Addition of two floating-point numbers follows rules in add (IEEE mode) or add (ALT mode).

Format:
[(pred)] add[.cmod] (exec_size) dst src0 src1

Programming Notes

Use a source modifier with add to implement subtraction.

Syntax

```
[(pred)] add[.cmod] (exec_size) reg reg reg [(pred)] add[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] + src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*B,*W,*D	F
F	F
DF	DF
HF	HF
*B,*W,*D	HF
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q
HF, F	HF, F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM')

add - Addition

		Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Addition with Carry

addc - Addition with Carry

Source: Eulsa

Length Bias: 4

The addc instruction performs component-wise addition of src0 and src1 and stores the results in dst; it also stores the carry into acc.
If the operation produces a carry out, 0x00000001 is stored in acc, else 0x00000000 is stored in acc.

Format:
[(pred)] addc[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

```
[(pred)] addc[.cmod] (exec_size) reg reg reg [(pred)] addc[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] + src1.chan[n]; acc.chan[n] = carry(src0.chan[n] + src1.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Arithmetic Shift Right

asr - Arithmetic Shift Right

Source: Eulsa

Length Bias: 4

Perform component-wise arithmetic right shift of the bits in src0 by the shift count indicated in src1, storing the results in dst. If src0 has a signed type, insert copies of src0's sign bit in the number of MSBs indicated by the shift count. Otherwise insert 0 bits.
The shift count is taken from the low five bits of src1, regardless of the src1 type and treated as an unsigned integer in the range 0 to 31.
In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.
For positive values, this operation is src0 / 2shiftCount and for negative values, this operation is src0 / 2shiftCount - 1.

Format:
[(pred)] asr[.cmod] (exec_size) dst src0 src1

Programming Notes

If src0 is -1, the result is -1 regardless of the shift count.

For unsigned src0 types, asr and shr produce the same result.

Syntax

```
[(pred)] asr[.cmod] (exec_size) reg reg reg [(pred)] asr[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) {: shiftCnt = src1.chan[n] & 0x1F; // Always use low 5 bits for shift count shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F if (src0.chan[n] >= 0) { dst.chan[n] = src0.chan[n] » shiftCnt; } else { int maskLSB = pow(2, shiftCnt) - 1; if ( maskLSB & src0.chan[n] == 0 ) { dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt); } else { dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt) - 1; } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q

asr - Arithmetic Shift Right

*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Average

avg - Average

Source: Eulsa

Length Bias: 4

The avg instruction performs component-wise integer average of src0 and src1 and stores the results in dst. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic right shift to this intermediate value.

Format:
The avg instruction performs component-wise integer average of src0 and src1 and stores the results in dst. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic right shift to this intermediate value.

Syntax

```
[(pred)] avg[.cmod] (exec_size) reg reg reg [(pred)] avg[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = (src0.chan[n] + src1.chan[n] + 1) » 1; // Use arithmetic shift right. } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Bit Field Extract

bfe - Bit Field Extract

Source: Eulsa

Length Bias: 4

Component-wise extract a bit field from src2 using the bit field width from src0 and the bit field offset from src1. Store the extracted bit field value in the low bits of dst and sign extend (if D type) or zero extend (if UD type).

The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f.

If width is zero, the result is zero.

If offset + width > 32 then the extracted bit field is bits offset to 31 of src2, extracting only 32 - offset bits, less than width as the bit field cannot extend past the MSB of the source value. Otherwise extract width bits extending from bit positions offset to offset + width - 1.

Format:

[(pred)] bfe (exec_size) dst src0 src1 src2

Restriction

Restriction : No accumulator access, implicit or explicit.

Restriction : All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] bfe (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD
width = src0.chan[n][4:0]; UD offset = src1.chan[n][4:0]; if ( width == 0 ) {
dst.chan[n] = 0x00000000; } else if ( (width + offset) < 32 ) { dst.chan[n] =
src2.chan[n] « (32 - width - offset); if (src2 is signed) { dst.chan[n] =
dst.chan[n] » (32 - width); // pad sign bit of dst.chan } else { dst.chan[n] =
dst.chan[n] » (32 - width); // pad 0 } } else { if ( src2 is signed ) {
dst.chan[n] = src2.chan[n] » offset; // pad sign bit } else { dst.chan[n] =
src2.chan[n] » offset; // pad 0 } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:126	**Reserved**

bfe - Bit Field Extract

	Format:	MBZ
125:106	**Source 2**	
	Format:	EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
105	**Reserved**	
	Format:	MBZ
104:85	**Source 1**	
	Format:	EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
84	**Reserved**	
	Format:	MBZ
83:64	**Source 0**	
	Format:	EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
63:56	**Destination Register Number**	
	Format:	DstRegNum
55:53	**Destination Subregister Number**	
	Format:	DstSubRegNum[2:0]
52:49	**Destination Channel Enable**	
	Format:	ChanEn[4]
	Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group	
48	**Reserved**	
	Format:	MBZ
47	**NibCtrl**	
	Format:	NibCtrl
46	**Reserved**	
	Format:	MBZ
45:44	**Destination Data Type**	
	This field contains the data type for the destination	

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

bfe - Bit Field Extract

48:42 Reserved

Format:	MBZ

43:42 Source Data Type

This field contains the data type for all three sources

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

41:40 Source 2 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

39:38 Source 1 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

41:36 Reserved

Exists If:	//([Property[Source Modifier]=='false')
Format:	MBZ

37:36 Source 0 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

35 Reserved

Format:	MBZ

34 Reserved

Format:	MBZ

34 Flag Register Number

This field contains the flag register number for instructions with a non-zero Conditional Modifier.

33 Flag Subregister Number

This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.

32 Destination Register File

Value	Name
0	GRF
1	MRF

bfe - Bit Field Extract			
	32	**Reserved**	
		Format:	MBZ
	31:0	**Header**	
		Format:	EU_INSTRUCTION_HEADER

Bit Field Insert 1

bfi1 - Bit Field Insert 1

Source: Eulsa

Length Bias: 4

The bfi1 instruction is the first instruction in a two-instruction macro for bfi (Bit Field Insert).
The bfi1 instruction component-wise generates mask with control from src0 and src1 and stores the results in dst. The mask is used in the bfi2 instruction to generate the final result of bfi.
Create a bit mask corresponding to the bit field width and offset in src0 and src1. Store the bit mask in dst. The mask has all bits in the bit field set to 1 and all other bits as 0.
The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f.
If width is zero, the result is zero.
The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value.

```
bfi dst src0 src1 src2 src3
// Translates to these two instructions:
bfi1 dst src0 src1
bfi2 dst dst src2 src3
```

Format:

```
[(pred)] bfi1 (exec_size) dst src0 src1
```

Programming Notes

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] bfi1 (exec_size) reg reg reg [(pred)] bfi1 (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD width = src0.chan[n][4:0]; UD offset = src1.chan[n][4:0]; dst = ((1 « width) - 1) « offset; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description

bfi1 - Bit Field Insert 1

0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Bit Field Insert 2

bfi2 - Bit Field Insert 2

Source: Eulsa

Length Bias: 4

The bfi2 instruction is the second instruction in a two-instruction macro for bfi (Bit Field Insert).
The bfi2 instruction component-wise performs the bitfield insert operation on src1 and src2 based on the mask in src0.
Use the mask in src0 to take a bit field value from the low bits of src1 and combine it with the value from src2 (so src2 provides all bits other than those masked out and replaced by the bit field value). Store the result in dst.
The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value.
bfi dst src0 src1 src2 src3
// Translates to these two instructions:
bfi1 dst src0 src1
bfi2 dst dst src2 src3

Format:
[(pred)] bfi2 (exec_size) dst src0 src1 src2

Restriction

Restriction : No accumulator access, implicit or explicit.

Restriction : All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] bfi2 (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD
offset = LZD(reverse(src0.chan[n]))-1; // offset is the number of LSB zero bits
below the bit mask which has all 1s. // width (implied by the logic) is the
number of 1 bits in the mask value, which should be all 1s. dst.chan[n] =
((src1.chan[n] « offset) & src0.chan[n]) | (src2.chan[n] & ! src0.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group
	48	**Reserved** Format: MBZ
	47	**NibCtrl** Format: NibCtrl
	46	**Reserved** Format: MBZ
	45:44	**Destination Data Type** This field contains the data type for the destination Value / Name: 00b — Single Precision Float 01b — DWord 10b — Unsigned DWord

bfi2 - Bit Field Insert 2

	11b	Double Precision Float

48:42 Reserved

Format:	MBZ

43:42 Source Data Type

This field contains the data type for all three sources

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

41:40 Source 2 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

39:38 Source 1 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

41:36 Reserved

Exists If:	//([Property[Source Modifier]=='false')
Format:	MBZ

37:36 Source 0 Modifier

Exists If:	//([Property[Source Modifier]=='true')
Format:	**SrcMod**

35 Reserved

Format:	MBZ

34 Reserved

Format:	MBZ

34 Flag Register Number

This field contains the flag register number for instructions with a non-zero Conditional Modifier.

33 Flag Subregister Number

This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.

32 Destination Register File

Value	Name
0	GRF
1	MRF

bfi2 - Bit Field Insert 2		
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Bit Field Reverse

bfrev - Bit Field Reverse

Source: Eulsa

Length Bias: 4

The bfrev instruction component-wise reverses all the bits in src0 and stores the results in dst.

Format:
[(pred)] bfrev (exec_size) dst src0

Restriction

Restriction : No accumulator access, implicit or explicit.

Syntax

```
[(pred)] bfrev (exec_size) reg reg [(pred)] bfrev (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { for ( idx = 0; idx < 32; idx++ ) { dst.chan[n][idx] = src0.chan[n][31-idx]; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Branch Converging

brc - Branch Converging

Source: Eulsa

Length Bias: 4

Description

The brc instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if all channels are branched away.
UIP should reference the instruction where all channels are expected to come together. JIP should reference the end of the innermost conditional block.

In GEN binary, JIP and UIP use locations src1 and src0 respectively when immediate and location src0 when reg64, where reg64 is accessed as paired DWord (regioning being <2;2,1>). dst must be IP. When the offsets are immediate, src0 regfile must be immediate.

Format:
[(pred)] brc (exec_size) JIP UIP

Restriction

Restriction : A brc instruction cannot use the Switch instruction option.

Syntax

```
[(pred)] brc (exec_size) imm32 imm32 [(pred)] brc (exec_size) reg64
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn[n] ) { PcIP[n] = IP + UIP;
} else { PcIP[n] = IP + 1; } } if ( all PcIP != IP + 1 ) { // for all channels
Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types
D

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.

brc - Branch Converging		
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Branch Diverging

brd - Branch Diverging

Source: Eulsa

Length Bias: 4

Description

The brd instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if any channels are branched away.

In GEN binary, JIP is at location src1 when immediate and at location src0 when reg32, where reg32 is accessed as a scalar DWord. The ip register must be used (for example, by the assembler) as dst.

Format:
[(pred)] brd (exec_size) JIP

Restriction

Restriction : A brd instruction cannot use the Switch instruction option.

Syntax

```
[(pred)] brd (exec_size) imm32 [(pred)] brd (exec_size) reg32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn[n] ) { PcIP[n] = IP + JIP;
} else { PcIP[n] = IP + 1; } } if ( any PcIP == ExIP + JIP ) { // any channel
Jump(ExIP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types
D

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: SrcType
	90:89	**Src1.RegFile** Format: RegFile

brd - Branch Diverging

	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Break

break - Break

Source: Eulsa

Length Bias: 4

Description

The break instruction is used to early-out from the inner most loop, or early out from the inner most switch block.

When used in a loop, upon execution, the break instruction terminates the loop for all execution channels enabled. If all the enabled channels hit the break instruction, jump to the instruction referenced by JIP. JIP should be the offset to the end of the inner most conditional or loop block, UIP should be the offset to the while instruction of the loop block.

If SPF is ON, the UIP must be used to update IP; JIP is not used in this case

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.

Format:
[(pred)] break (exec_size) JIP UIP

Syntax

```
[(pred)] break (exec_size) imm32 imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) {
PcIP[n] = IP + UIP; else { PcIP[n] = IP + 1; } } if ( PcIP != (IP + 1) ) { // all
channels Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Byte Scaled Read MSD

MSD2R_BS - Byte Scaled Read MSD

Source: DataPort 2
Length Bias: 1
Family: Scaled R/W
Group: Byte Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A32_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 04h Format: Opcode Byte Scattered Read message
	14:12	**Reserved** Format: MBZ Ignored
	11:10	**Data Elements** Format: MDC_DS Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Format: MBZ Ignored
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Sideband Scaled Offset** Format: MDC_A32_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

Byte Scaled Write MSD

MSD2W_BS - Byte Scaled Write MSD

Source:	DataPort 2
Length Bias:	1
Family:	Scaled R/W
Group:	Byte Scaled R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A32_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 0Ch Format: Opcode Byte Scattered Write message
	14:12	**Reserved** Format: MBZ Ignored
	11:10	**Data Elements** Format: MDC_DS Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Format: MBZ Ignored
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Sideband Scaled Offset** Format: MDC_A32_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

Byte Scattered Read MSD

MSD0R_BS - Byte Scattered Read MSD

Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 04h Format: Opcode Byte Scattered Read message
	13	**Reserved** Format: MBZ Ignored
	12	**Reserved** Format: MBZ Ignored
	11:10	**Data Elements** Format: MDC_DS Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Format: MBZ Ignored

MSD0R_BS - Byte Scattered Read MSD		
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Byte Scattered Write MSD

MSD0W_BS - Byte Scattered Write MSD

Source:	DataPort 0
Length Bias:	1
Family:	Scattered R/W
Group:	Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Byte Scattered Write message
	13:12	**Reserved** Format: MBZ Ignored
	11:10	**Data Elements** Format: MDC_DS Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Format: MBZ Ignored
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)

MSD0W_BS - Byte Scattered Write MSD		
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Call

call - Call

Source: Eulsa

Length Bias: 4

Description

The call instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the call instruction. If none of the channels jump into the subroutine, the call instruction is treated as a nop.

In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register.

When SPF is on, the predication control must be scalar.

The following section describes JIP, the jump offset.

JIP can be an immediate or register value. When a jump occurs, this value is added to IP pre-increment. In GEN binary, JIP is at location src1 and src0 must be null. The GRF register must be put (for example, by the assembler) at dst location.

Format: [(pred)] call (exec_size) dst JIP

Format: [(pred)] call (exec_size) dst JIP

Restriction

Restriction : The call instruction must have DWord source and destination type, and the destination must be QWord aligned.

Restriction : A call instruction must target an instruction with Switch set; in addition, the instruction following the call must also have Switch set.

Syntax

```
[(pred)] call (exec_size) reg imm32 [(pred)] call (exec_size) reg reg32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if (WrEn.chan[n] ) { PcIP[n] = IP + JIP; CallMask[n] = 1; } else { PcIP[n] = IP + 1; CallMask[n] = 0; } } if ( PcIP[n] != (IP + 1) ) { // any channel jumped dst.chan[0] = IP + 1; dst.chan[1] = CallMask; Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Dst Types
D, UD

DWord	Bit	Description
0..3	127:96	**JIP**

call - Call

		Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: SrcType
	90:89	**Src1.RegFile** Format: RegFile
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Call Absolute

calla - Call Absolute

Source: Eulsa

Length Bias: 4

The calla instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the calla instruction. If none of the channels jump into the subroutine, the calla instruction is treated as a nop.

In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register.

If SPF is ON, none of the PcIP are updated.

When SPF is on, the predication control must be scalar.

The difference between calla and call is that calla uses JIP as the IP value rather than adding it to the IP value.

Format:

[(pred)] calla (exec_size) dst JIP

Restriction

Restriction : The calla instruction must have DWord source and destination type, and the destination must be QWord-aligned.

Restriction : The src0 regioning control must be <2;2,1>,

Syntax

```
[(pred)] calla (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) {
PcIP[n] = JIP; CallMask[n] = 1; } else { PcIP[n] = IP + 1; CallMask[n] = 0; } }
if ( PcIP[n] != (IP + 1) ) { // any channel jumped dst.chan[0] = IP + 1;
dst.chan[1] = CallMask; Jump(JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Dst Types
D, UD

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**

calla - Call Absolute

	94:91	**Src1.SrcType** Format: **SrcType**
	90:89	**Src1.RegFile** Format: **RegFile**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Compare

cmp - Compare

Source: Eulsa

Length Bias: 4

The cmp instruction performs component-wise comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional code (excluding NS signal) based on the conditional modifier, and storing the conditional bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results.

A conditional modifier must be specified; the conditional modifier field cannot be 0000b. The comparison does not use the NS (NaN source) signals, as described in the Creating Conditional Flags section. Accordingly the conditional modifier should not be .u (unordered).

For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst.

When any source type is floating-point, the cmp instruction obeys the rules described in the tables in the Floating Point Modes section of the Data Types chapter.

Format:

[(pred)] cmp[.cmod] (exec_size) dst src0 src1

Syntax

```
[(pred)] cmp[.cmod] (exec_size) reg reg reg [(pred)] cmp[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if ( WrEn.chan[n] ) { results[n] = src0.chan[n] - src1.chan[n]; bitMask[n] = Condition(results[n]); dst.chan[n] = bitMask[n]; // All bits for dst channel } } flag# = bitMask;
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*B,*W,*D	F
F	F
DF	DF
HF	HF
*B,*W,*D	HF

cmp - Compare

*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q
HF, F	HF, F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Compare NaN

cmpn - Compare NaN

Source: Eulsa

Length Bias: 4

The cmpn instruction performs component-wise special-NaN comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional signals including NS based on the conditional modifier, and storing the conditional flag bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results.

A conditional modifier must be specified; the conditional modifier field cannot be 0000b. More information about the conditional signals used is in the Creating Conditional Flags section.

For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst.

Min/Max instructions use cmpn to select the destination from the input sources (see the Min Max of Floating Point Numbers section for details).

Format:

[(pred)] cmpn[.cmod] (exec_size) dst src0 src1

Restriction

Restriction :

.l and .ge are the only two conditional modifiers are supported for this instruction.

Syntax

```
[(pred)] cmpn[.cmod] (exec_size) reg reg reg [(pred)] cmpn[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if ( WrEn.chan[n] ) { results[n] = src0.chan[n] - src1.chan[n]; bitMask[n] = ConditionNaN(results[n]); dst.chan[n][0] = bitMask[n]; // All bits for dst channel } } flag# = bitMask;
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*B,*W,*D	F
F	F

cmpn - Compare NaN

DF	DF
HF	HF
F, HF	F, HF

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Conditional Select

csel - Conditional Select

Source: Eulsa

Length Bias: 4

The csel instruction selectively moves components in src0 or src1 to the dst based on the result of the compare of src2 with zero. If the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst. The csel instruction provides the function of a cmp followed by sel. The instruction must not be used if cmpn is required. The instruction does not update the flag register.
The comparison follows the same rule as cmp instruction for that data type.

Format:
csel (exec_size) dst src0 src1 src2

Syntax

```
csel[.cmod] (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if (
EMask.chan[n] ) { result[n] = src2.chan[n] - 0; bitMask[n] =
Condition(result[n]); if (bitMask[n] = 1) { dst.chan[n] = src0.chan[n]; } else {
dst.chan[n] = src1.chan[n]; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

csel - Conditional Select

	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group
	48	**Reserved** Format: MBZ
	47	**NibCtrl** Format: NibCtrl
	46	**Reserved** Format: MBZ
	45:44	**Destination Data Type** This field contains the data type for the destination

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

	48:42	**Reserved** Format: MBZ
	43:42	**Source Data Type** This field contains the data type for all three sources

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

csel - Conditional Select

	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	34	**Flag Register Number** This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Destination Register File** Value: 0 — Name: GRF Value: 1 — Name: MRF
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Conditional Send Message

sendc - Conditional Send Message

Source: Eulsa

Length Bias: 4

The sendc instruction has the same behavior as the send instruction except the following.
sendc first checks the dependent threads inside the Thread Dependency Register. There are up to 8 dependent threads in the TDR register. The sendc instruction executes only when all the dependent threads in the TDR register are retired.
Wait for dependencies in the TDR Register to clear, then send a message stored in registers starting at src to a shared function identified by exdesc along with control from desc with a general register writeback location at dst.

Format:
[(pred)] sendc (exec_size) dst src0 exdesc desc

Restriction

Restriction : The sendc instruction has the same restrictions as the send instruction.

Syntax

```
[(pred)] sendc (exec_size) reg reg imm32 reg32a [(pred)] sendc (exec_size) reg reg imm32 imm32
```

Pseudocode

```
if ( TDR[7] ... || TDR[2] || TDR[1] || TDR[0] ) { wait; } Evaluate(WrEn); MsgChEnable = WrEn; SourceReg = src0.RegNum; MessageEnqueue(MsgChEnable, ResponseReg, SourceReg, desc, exdesc);
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95:89	**Flags** Format: EU_INSTRUCTION_FLAGS
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

sendc - Conditional Send Message

	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Shared Function ID (SFID)** Format: SFID
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Conditional Split Send Message

sendsc - Conditional Split Send Message

Source: Eulsa

Length Bias: 4

The sendsc instruction has the same behavior as the sends instruction except the following.
sendsc first checks the dependent threads inside the Thread Dependency Register. There are up to 8 dependent threads in the TDR register. The sendsc instruction executes only when all the dependent threads in the TDR register are retired.

Wait for dependencies in the TDR Register to clear, then send a message stored in GRF locations starting at <src0> followed by <src1> to a shared function identified by <ex_desc> along with control from <desc> and <ex_desc> with a GRF writeback location at <dest>.

Format:
[(pred)] sendsc (exec_size) <dest> <src0> <src1> <ex_desc> <desc>

Restriction

Restriction : The sendsc instruction has the same restrictions as the sends instruction.

Syntax

```
[(pred)] sendsc (exec_size) reg greg reg imm32 reg32a
 [(pred)] sendsc (exec_size) reg greg reg imm32 imm32
 [(pred)] sendsc (exec_size) reg greg reg reg32a imm32
 [(pred)] sendsc (exec_size) reg greg reg reg32a reg32a
```

Pseudocode

```
if ( TDR[7] ... || TDR[2] || TDR[1] || TDR[0] ) { wait; } Evaluate(WrEn);
<MsgChEnable> = WrEn; <SourceReg0> = <src0>.RegNum; <SourceReg1> = <src1>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg0>,
<SourceReg1>,<ex_desc>, <dest>);
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95:80	**ExDesc[31:16]** Format: ExtMsgDescpt[31:16]
	79	**Source 0 Addressing Mode** Format: AddrMode

sendsc - Conditional Split Send Message

	78	**Reserved** Exists If: ([Source 0 Addressing Mode]=='Direct') Format: MBZ
	78	**Source 0 Address Immediate Sign [9]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[9]
	77	**SelReg32Desc**
	76:73	**Source 0 Address Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Indirect')
	76:69	**Source 0 Register Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	72:68	**Source 0 Address Immediate [8:4]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[8:4]
	68	**Source 0 Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	67:64	**ExDesc[9:6]** Format: **ExtMsgDescpt[9:6]**
	63	**Destination Addressing Mode** Format: **AddrMode**
	62	**Destination Address Immediate Sign [9]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[9]
	62	**Reserved** Exists If: ([Destination Addressing Mode]=='Direct') Format: MBZ
	61	**Reserved** Format: MBZ
	60:57	**Destination Address Subregister Number** Exists If: ([Destination Addressing Mode]=='Indirect')
	60:53	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct')
	56:52	**Destination Address Immediate [8:4]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[8:4]

sendsc - Conditional Split Send Message

	52	**Destination Subregister Number [4]** Exists If: ([Destination Addressing Mode]=='Direct')
	51:44	**Source 1 Register Number**
	43:41	**Reserved** Format: MBZ
	40:37	**Destination Type**
	36	**Source 1 Register File** Format: RegFile[0]
	35	**Destination Register File** Format: RegFile[0]
	34	**MaskCtrl**
	33:32	**Flag Register Number/Subregister Number**
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Shared Function ID (SFID)** Format: SFID
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Constant Cache Dword Scattered Read MSD

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD

Source:	Read-Only DataPort
Length Bias:	1
Family:	Scattered R/W
Group:	DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 03h Format: Opcode Dword Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**Reserved** Format: MBZ Ignored
	11:10	**Reserved** Format: MBZ Ignored
	9	**Legacy SIMD Mode** Default Value: 1h Format: Opcode Must be set for compatibility.

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD		
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Constant Cache Oword Block Read MSD

MSD_CC_OWB - Constant Cache Oword Block Read MSD

Source: Read-Only DataPort
Length Bias: 1
Family: Block R/W
Group: OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 00h Format: Opcode Oword Block Read Constant Cache message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Constant Cache Oword Dual Block Read MSD

MSD_CC_OWDB - Constant Cache Oword Dual Block Read MSD

Source: Read-Only DataPort
Length Bias: 1
Family: Block R/W
Group: OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 02h Format: Opcode Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:10	**Reserved** Format: MBZ Ignored
	9:8	**Data Elements** Format: MDC_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Constant Cache Oword Unaligned Block Read MSD

MSD_CC_OWUB - Constant Cache Oword Unaligned Block Read MSD

Source:	Read-Only DataPort
Length Bias:	1
Family:	Block R/W
Group:	OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 01h Format: Opcode Oword Unaligned Block Read Constant Cache message
	13:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Continue

cont - Continue

Source: Eulsa

Length Bias: 4

Description

The cont instruction disables execution for the subset of channels for the remainder of the current loop iteration. Channels remain disabled until right before the while instuction or right before the condition check code block for the while instruction. If all enabled channels hit this instruction, jump to the instruction referenced by JIP where execution continues.

UIP should always reference the loop's associated while instruction. JIP should point to the last instruction of the inner most conditional block if the cont instruction is inside a conditional block. In case of the break instruction directly under the loop, the JIP and the UIP are the same.

If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.

Format:
[(pred)] cont (exec_size) JIP UIP

Restriction

Restriction : The execution size must be the same for the while, break, and cont instructions of the same code block.

Syntax

```
[(pred)] cont (exec_size) imm32 imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) { if ( PMask[n] ) { // PMask is for all channels enabled for the cont instruction. PcIP[n] = IP + UIP; } else { PcIP[n] = IP + 1; } } } for ( n = exec_size; n < 32; n++ ) { PcIP[n] = IP + 1; } if ( PcIP != (IP + 1) ) { // all channels true Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.

cont - Continue		
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Count Bits Set

cbit - Count Bits Set

Source: Eulsa

Length Bias: 4

The cbit instruction counts component-wise the total bits set in src0 and stores the resulting counts in dst.

Format:
[(pred)] cbit (exec_size) dst src0

Restriction

Restriction : No accumulator access, implicit or explicit.

Syntax

```
[(pred)] cbit (exec_size) reg reg [(pred)] cbit (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt
= 0; UD val = src0.chan[n]; while ( val ) { if ( val & 1 ) { cnt ++; } val = val
» 1; } dst.chan[n] = cnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UB, UW, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Dot Product 2

dp2 - Dot Product 2

Source: Eulsa

Length Bias: 4

The dp2 instruction performs a two-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every third and fourth element of src0 (post-source-swizzle if present) are not involved in the computation.

The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements.

The dp4 instruction includes all four elements of each vector in the dot product. The dp3 instruction includes the first three elements of each vector in the dot product.

Format:

[(pred)] dp2[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Execution size cannot be less than 4.

Restriction : Horizontal strides must be 1.

Restriction : Source operands cannot be accumulators.

Syntax

```
[(pred)] dp2[.cmod] (exec_size) reg reg reg [(pred)] dp2[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

dp2 - Dot Product 2		
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Dot Product 3

dp3 - Dot Product 3

Source: Eulsa

Length Bias: 4

The dp3 instruction performs a three-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every fourth element of src0 (post-source-swizzle if present) is not involved in the computation.

The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements.

The dp4 instruction includes all four elements of each vector in the dot product. The dp2 instruction includes the first two elements of each vector in the dot product.

Format:

[(pred)] dp3[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Execution size cannot be less than 4.

Restriction : Horizontal strides must be 1.

Restriction : Source operands cannot be accumulators.

Syntax

```
[(pred)] dp3[.cmod] (exec_size) reg reg reg [(pred)] dp3[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

dp3 - Dot Product 3

	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Dot Product 4

dp4 - Dot Product 4

Source: Eulsa

Length Bias: 4

The dp4 instruction performs a four-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst.
The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements.

Format:
[(pred)] dp4[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Execution size cannot be less than 4.

Restriction : Horizontal strides must be 1.

Restriction : Source operands cannot be accumulators.

Syntax

```
[(pred)] dp4[.cmod] (exec_size) reg reg reg [(pred)] dp4[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2] + src0.chan[n+3] * src1.chan[n+3]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM

dp4 - Dot Product 4		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Dot Product Homogeneous

dph - Dot Product Homogeneous

Source: Eulsa

Length Bias: 4

The dph instruction performs a four-wide homogeneous dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every fourth element of src0 (post-source-swizzle if present) is forced to 1.0f.
Use the dp4 instruction to do a four-wide dot product that includes all elements of src0 and src1.

Format:
[(pred)] dph[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Execution size cannot be less than 4.

Restriction : Horizontal strides must be 1.

Restriction : Source operands cannot be accumulators.

Syntax

```
[(pred)] dph[.cmod] (exec_size) reg reg reg [(pred)] dph[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2] + src1.chan[n+3]; // Use 1.0f in place of src0.chan[n+3]. if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM

dph - Dot Product Homogeneous		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Dword Atomic Counter Binary with Return Data Operation MSD

MSD1R_DWAC2 - Dword Atomic Counter Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2RS Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Atomic Counter Binary Write Only Operation MSD

MSD1W_DWAC2 - Dword Atomic Counter Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2RS Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Atomic Counter Unary with Return Data Operation MSD

MSD1R_DWAC1 - Dword Atomic Counter Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2RS Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Atomic Counter Unary Write Only Operation MSD

MSD1W_DWAC1 - Dword Atomic Counter Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2RS Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Scattered Read MSD

MSD0R_DWS - Dword Scattered Read MSD

Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 03h Format: Opcode Dword Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**Reserved** Format: MBZ Ignored
	11:10	**Reserved** Format: MBZ Ignored
	9	**Legacy SIMD Mode** Default Value: 1h Format: Opcode Must be set for compatibility.

MSD0R_DWS - Dword Scattered Read MSD		
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Dword Scattered Write MSD

MSD0W_DWS - Dword Scattered Write MSD

Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Bh Format: Opcode Dword Scattered Write message
	13:12	**Reserved** Format: MBZ Ignored
	11:10	**Reserved** Format: MBZ Ignored
	9	**Legacy SIMD Mode** Default Value: 1h Format: Opcode Must be set for compatibility.
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)

MSD0W_DWS - Dword Scattered Write MSD		
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD

MSD1R_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD

MSD1W_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD

MSD1R_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD

MSD1W_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD

MSD1W_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Typed Atomic Operation
Group: Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP3S Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP3S Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD

MSD1W_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD

MSD1R_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD

MSD1W_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD

MSD1R_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

MSD1W_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Float Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD		
	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Format: MBZ Ignored
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWTAI2 - Dword Typed Atomic Integer Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Binary Write Only Operation MSD

MSD1W_DWTAI2 - Dword Typed Atomic Integer Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWTAI3 - Dword Typed Atomic Integer Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP3S Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWTAI3 - Dword Typed Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP3S Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWTAI1 - Dword Typed Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Unary Write Only Operation MSD

MSD1W_DWTAI1 - Dword Typed Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Format: MDC_SG2 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Untyped Atomic Float Binary with Return Data Operation MSD

MSD1R_DWAF2 - Dword Untyped Atomic Float Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_DWAF2 - Dword Untyped Atomic Float Binary with Return Data Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Float Binary Write Only Operation MSD

MSD1W_DWAF2 - Dword Untyped Atomic Float Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_DWAF2 - Dword Untyped Atomic Float Binary Write Only Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP2 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Float Trinary with Return Data Operation MSD

MSD1R_DWAF3 - Dword Untyped Atomic Float Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1R_DWAF3 - Dword Untyped Atomic Float Trinary with Return Data Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Float Trinary Write Only Operation MSD

MSD1W_DWAF3 - Dword Untyped Atomic Float Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Float Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10	**Reserved** Format: MBZ Ignored

MSD1W_DWAF3 - Dword Untyped Atomic Float Trinary Write Only Operation MSD

	9:8	**Atomic Float Operation** Format: MDC_FOP3 Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWAI2 - Dword Untyped Atomic Integer Binary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_DWAI2 - Dword Untyped Atomic Integer Binary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP2 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWAI3 - Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWAI3 - Dword Untyped Atomic Integer Trinary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP3 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWAI1 - Dword Untyped Atomic Integer Unary with Return Data Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_DWAI1 - Dword Untyped Atomic Integer Unary Write Only Operation MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP1 Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Else

else - Else

Source: Eulsa

Length Bias: 4

The else instruction is an optional statement within an if/else/endif block of code. It restricts execution within the else/endif portion to the opposite set of channels enabled under the if/else portion. Channels which were inactive prior to entering the if/endif block remain inactive throughout the entire block.

All enabled channels upon arriving the else instruction will be redirected to the matching endif. If all channels are redirected (by else or before else), a relative jump is performed to the location specified by <JIP>. The jump target should be the matching endif instruction for that conditional block.

The following table describes the 32-bit <JIP>. In GEN binary, <JIP> is at location <src1> and must be of type D (signed dword integer). <JIP> must be an immediate operand, it is a signed 32-bit number and is intended to be forward referencing. This value is added to IP pre-increment.

If the <branch_ctrl> bit is set, then the <JIP> points to the first join instruction within the else block and <UIP> points to the endif instruction. If the <branch_ctrl> bit is not set, <JIP> and <UIP>, both point to endif.

Format:

else (<exec_size>) <JIP> <UIP> <branch_ctrl>

Programming Notes

If all channels are redirected (by else or before else), relative jump is performed to the location specified by <JIP> + 1.

Restriction

Restriction : Predication is not allowed.

Restriction : The execution size must be the same for the if, else, and endif instructions of the same code block.

Restriction : The branch_ctrl must be set equal to (JIP != UIP).

Syntax

```
else (<exec_size>) imm32 imm32 branch_ctrl
```

Pseudocode

```
Evaluate(WrEn); for (n = 0; n < 32; n++) { if (WrEn.channel[n] == 1 ||
branch_ctrl) { PcIP[n] = IP + JIP; } else { PcIP[n] = IP + UIP; } } if (PcIP !=
(IP+1)) { // for all channels Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP**

else - Else

		Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

End If

endif - End If

Source: Eulsa

Length Bias: 4

Description

The endif instruction terminates an if/else/endif block of code. It restores execution to the channels that were active prior to the if/else/endif block.
The endif instruction is also used to hop out of nested conditionals by jumping to the end of the next outer conditional block when all channels are disabled.

The following table describes the 32-bit JIP. In GEN binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand, it is a signed 32-bit number. This value is added to IP pre-increment.

Format:
endif JIP

Restriction

Restriction : Predication is not allowed.

Restriction : The execution size must be the same for the if, else, and endif instructions of the same code block.

Syntax

```
endif (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn); if ( WrEn == 0 ) { // all channels false Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: SrcType
	90:89	**Src1.RegFile** Format: RegFile

endif - End If

	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Extended Math Function

math - Extended Math Function

Source:	Eulsa
Length Bias:	4

The math instruction performs extended math function on the components in src0, or src0 and src1, and write the output to the channels of dst. The type of extended math function are based on the FC[3:0] encoding in the table below.

Format:
[(pred)] math (exec_size) dst src0 src1 <FC>

Restriction
Restriction : Accumulator access is allowed only for ieee macro functions.
Restriction : The math instruction does not support indirect addressing modes.
Restriction : The only supported rounding mode for math instruction is Round to Nearest Even.
Restriction : INT DIV function does not support SIMD16. INT DIV function does not support simulataneous write to two registers. INT DIV function does not support source modifiers.
Restriction : The FDIV function is not supported in ALT_MODE.
Restriction : The math instruction can use GRF or immediates as source. The source formats for immediates must be one of the source formats supported by math operation.
Restriction : DepCtrl must not be used.
Restriction : The math instruction must use GRF as destination.
Restriction : The supported regioning mode for math instruction is align1 and align16. The following restrictions apply for align1 mode: Scalar source is supported. Source and destination horizontal stride must be the same. Regioning must ensure Src.Vstride = Src.Width * Src.Hstride . Source and destination offset must be the same, except the case of scalar source.
Restriction : Half-float denorms are always retained.
Restriction : Math Operation rules when float and half-floats are mixed between source or between source and destination operands. The half-float operand must be interleaved in the register for align1 and the source and destination register offset must be be the same to DW granularity. For align16, the half-float operand is allowed to be packed.
Restriction : The execution size must be no more than 8 when half-floats are used in source or destination operand.
Restriction : The source operand must not span two registers if the destination operand spans just one register Example: Case (a) // Allowed. The source must be strided by 2. the offset is allowed to select between lower/upper 16b math (8) r10:f r11.0<16;8,2>:hf 0x01

math - Extended Math Function

math (8) r10:f r11.1<16;8,2>:hf 0x01
math (8) r10:f r11.0<16;8,2>:hf r12.1<16;8,2>:hf 0x09
Case (b) // Allowed. The destination must be strided by 2. The offset is allowed to selecte between lower/upper 16b
math (8) r10.0<2>:hf r11.0<8;8,1>:f
math (8) r10.1<2>:hf r11.0<8;8,1>:f 0x01
math (8) r10.0<2>:hf r11.0<16;8,2>:hf r12.0<16;8,2>:hf 0x09
Case (c) // Allowed. Destination has stride of 2. The offset is allowed to select between uppoer/lower 16b
math (8) r10.0<2>:hf r11.0<8;8,1>:f r12.1<16;8,2>:hf 0x09
math (8) r10.1<2>:hf r11.1<16;8,2>:hf r12.0<8;8,1>:f 0x09
Case (d) // Not Allowed. Destination is half-float but is not interleaved.
math (8) r10.0<1>:hf r11.0<8;8,1>:f
Case (e) // Not Allowed. Source is half-float but not interleaved
math (8) r10.0<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf 0x09
Case (f) // Not Allowed. Source operand spans 2 registers while destination spans one register.
math (8) r83.8<1>:hf r12.4<4;4,1>:f null 0x02

Restriction : Math Operation rules when half-floats are used on both source and destination operands.
The execution size must be 8.
The half-float source must be packed or interleaved. When interleaving, both source and destination must be interleaved.
Example:
Case (a) // Allowed. The source and destination are packed or interleaved
math (8) r10.0:hf r11.0<8;8,1>:hf 0x01
math (8) r10.0<2>:hf r11.0<16;8,2>:hf 0x01
math (8) r10.8:hf r11.0<8;8,1>:hf 0x01
math (8) r10.8<2>:hf r11.0<16;8,2>:hf 0x01

Restriction : For one source math operations src1 must be NULL.

Syntax

```
[(pred)] math (exec_size) reg reg reg imm4
```

Pseudocode

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
    if (WrEn.channel[n] == 1) {
        switch FC[3:0] {
            case 1h:
                dst.channel[n] = rcp(src0.channel[n]);
            case 2h:
                dst.channel[n] = log(src0.channel[n]);
            case 3h:
                dst.channel[n] = exp(src0.channel[n]);
            case 4h:
                dst.channel[n] = sqrt(src0.channel[n]);
            case 5h:
                dst.channel[n] = rsq(src0.channel[n]);
            case 6h:
```

math - Extended Math Function

```
                dst.channel[n] = sin(src0.channel[n]);
            case 7h:
                dst.channel[n] = cos(src0.channel[n]);
            case 9h: // src0 / src1
                dst.channel[n] = fdiv(src0.channel[n], src1.channel[n]);
            case Ah:
                dst.channel[n] = pow(src0.channel[n], src1/channel[n]);
            case Bh: // src0 / src1
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = quotient;
                dst+1.channel[n] = remainder;
            case Ch:
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = quotient;
            case Dh:
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = remainder;
            }
        }
    }
}
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
F	F
D	D
UD	UD
F, HF	F, HF

DWord	Bit	Description
0..3	127:64	**RegSource** Format: EU_INSTRUCTION_SOURCES_REG_REG
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Function Control (FC)** Format: FC
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ

math - Extended Math Function		
	6:0	**Opcode** Format: EU_OPCODE

Find First Bit from LSB Side

fbl - Find First Bit from LSB Side

Source: Eulsa

Length Bias: 4

The fbl instruction counts component-wise the number of LSB 0 bits before the first 1 bit in src0, storing that number in dst.

Format:
[(pred)] fbl (exec_size) dst src0

Programming Notes

If src0 contains no 1 bits, store 0xFFFFFFFF in dst.

Restriction

Restriction : No accumulator access, implicit or explicit.

Syntax

```
[(pred)] fbl (exec_size) reg reg [(pred)] fbl (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt = 0; UD udScalar = src0.chan[n]; while ( (udScalar & 1) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar » 1; } if ( src0.chan[n] == 0x00000000 ) { dst.chan[n] = 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS

fbl - Find First Bit from LSB Side		
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Find First Bit from MSB Side

fbh - Find First Bit from MSB Side

Source: Eulsa

Length Bias: 4

If src0 is unsigned, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst.
If src0 is signed and positive, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst.
If src0 is signed and negative, the fbh instruction counts component-wise the leading ones from src0 and stores the resulting counts in dst.

Format:
[(pred)] fbh (exec_size) dst src0

Programming Notes

If src0 is zero, store 0xFFFFFFFF in dst.

If src0 is signed and is -1 (0xFFFFFFFF), store 0xFFFFFFFF in dst.

Restriction

Restriction : No accumulator access, implicit or explicit.

Syntax

```
[(pred)] fbh (exec_size) reg reg [(pred)] fbh (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt
= 0; if ( src0 is unsigned ) { UD udScalar = src0.chan[n]; while ( (udScalar & (1
« 31)) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar « 1; } if ( src0.chan[n]
== 0x00000000 ) { dst.chan[n] = 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } else
{ // src0 is signed. D dScalar = src0.chan[n]; bit cval = dScalar[31]; while
((dScalar & (1 « 31)) == cval && cnt != 32 ) { cnt ++; dScalar = dScalar « 1; }
if ( (src0.chan[n] == 0xFFFFFFFF) || (src0.chan[n] == 0x00000000) ) { dst.chan[n]
= 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
D, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG

fbh - Find First Bit from MSB Side		
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_IMM32**
	63:32	**Operand Controls** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Fraction

frc - Fraction

Source: Eulsa

Length Bias: 4

The frc instruction computes, component-wise, the truncate-to-minus-infinity fractional values of src0 and stores the results in dst. The results, in the range of [0.0, 1.0], are the fractional portion of the source data. The result is in the range [0.0, 1.0] irrespective of the rounding mode. Floating-point fraction computation follows the rules in the following tables, based on the current floating-point mode.

Format:
[(pred)] frc[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] frc[.cmod] (exec_size) reg reg [(pred)] frc[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) {
dst.chan[n] = src0.chan[n] - floor(src0.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Goto

goto - Goto

Source: Eulsa

Length Bias: 4

The goto instruction directs the instruction pointer to the offset specified by the UIP offset or to the next IP based on the BranchCtrl bit in the instruction. The active channels that are predicated on this instruction will take the IP + UIP path when BranchCtrl is set else the channels take IP + 1. The active channels that are not predicated on this instruction will be made inactive and waiting to be joined at the join IP. The join IP is IP + UIP when BranchCtrl is clear else it is the next IP.
When there are no active channels the instruction pointer will move to IP + JIP.
The goto instruction is used in conjunction with a join instruction. A goto deactivates some channels that are reactivated at some program-specified join instruction. See the join instruction for the activation rules.
The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence.
The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer).
If SPF is ON, none of the PcIP are updated.

Format:
[(pred)] goto (exec_size) JIP UIP BranchCtrl

Restriction

Restriction : Cannot have a goto in the body and the corresponding join in the function or the subroutine. Similarly the brd and brc.

Syntax

```
[(pred)] goto (exec_size) imm32 imm32 BranchCtrl
```

Pseudocode

Evaluate(WrEn);

```
 for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { // for the predicated
active channels if ( BranchCtrl ) { PcIP[n] = IP + UIP; } else { PcIP[n] = IP +
1; } } else { // join IP, for the active non predicated channels if ( BranchCtrl
) { PcIP[n] = IP + 1; } else { PcIP[n] = IP + UIP; } } } if ( BranchCtrl ) { //
if (PcIP != (IP + UIP) ) { // for all channels if (PcIP != (IP + 1) ) { // for
all channels Jump(IP + JIP); } else { Jump(IP + 1); } } else { Jump(IP + UIP); }
} else { // if (PcIP != (IP + 1) ) { // for all channels Jump(IP + JIP); } else {
Jump(IP + 1); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

goto - Goto

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

GPGPU_CSR_BASE_ADDRESS

GPGPU_CSR_BASE_ADDRESS

Source: BSpec

Length Bias: 2

The GPGPU_CSR_BASE_ADDRESS command sets the base pointers for EU and L3 to Context Save and Restore EU State and SLM for **GPGPU** mid-thread preemption.

Programming Notes

Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance. State and instruction caches are flushed on completion of the flush.

SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set prior to programming GPGPU_CSR_BASE_ADDRESS command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies for certain programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 04h GPGPU_CSR_BASE_ADDRESS Format: Opcode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length -2 Value / Name / Description: 1h / **[Default]** / Excludes DWord(0,1)

GPGPU_CSR_BASE_ADDRESS		
1..2	63:12	**GPGPU CSR Base Address** Format: GraphicsAddress[63:12] Specifies the 256K-byte aligned base address for GPGPU context GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11:0	**Reserved** Format: MBZ

GPGPU_WALKER

GPGPU_WALKER

Source: RenderCS

Length Bias: 2

Programming Notes

If the threads spawned by this command are required to observe memory writes performed by threads spawned from a previous command, software must precede this command with a command that performs a memory flush (e.g., MI_FLUSH).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h GPGPU_WALKER Format: OpCode
	23:16	**SubOpcode** Default Value: 05h GPGPU_WALKER SubOp Format: OpCode
	15:11	**Reserved** Format: MBZ
	10	**Indirect Parameter Enable** Format: Enable If set, the values in DW 7, 10, 12 are ignored and replaced by the current values of the corresponding GPGPU_xxx MMIO registers: • GPGPU_DISPATCHDIMX (instead of DW7) • GPGPU_DISPATCHDIMY (instead of DW10) • GPGPU_DISPATCHDIMZ (instead of DW12)
	9	**Reserved** Format: MBZ

GPGPU_WALKER

	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if **PredicateEnable** is set and the Predicate state bit is 0.
	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 0Dh / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31:17	**Reserved** Format: MBZ
	16:0	**Indirect Data Length** Format: U17 in bytes **Description** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. If Indirect Data is enabled, it is assumed that CURBE is not being used except for cross-thread constant data. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, the total size of Indirect data must be less than 63,488 (2048 URB lines - 64 lines for Interface Descriptors).

GPGPU_WALKER

3 — 31:6 — Indirect Data Start Address

Format:	IndirectObjectOffset[31:6]

This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address.**
Hardware ignores this field if indirect data is not present.
Alignment of this address depends on the mode of operation.
It is the 64-byte aligned address of the indirect data.

Value	Name	Description
[0 - 512MB]		(Bits 31:29 MBZ)

3 — 5:0 — Reserved

Format:	MBZ

4 — 31:30 — SIMD Size

This field determines the size of the payload and the number of bits of the execution mask that are expected. The kernel pointed to by the interface descriptor should match the SIMD declared here.

Value	Name	Description
0	SIMD8	8 LSBs of the execution mask are used
1	SIMD16	16 LSBs used in execution mask
2	SIMD32	32 bits of execution mask used

4 — 29:22 — Reserved

Format:	MBZ

4 — 21:16 — Thread Depth Counter Maximum

The maximum value of the thread depth counter. Since the counter starts at 0, the depth is this value + 1.
(Thread_Depth_Max+1)*(Thread_Height_Max+1)*(Thread_Width_Max+1) must equal **Number of Threads in GPGPU Thread Group** in the Interface Descriptor.

4 — 15:14 — Reserved

Format:	MBZ

4 — 13:8 — Thread Height Counter Maximum

Format:	U6-1

The maximum value of the thread height counter. The height is this value + 1.

4 — 7:6 — Reserved

Format:	MBZ

4 — 5:0 — Thread Width Counter Maximum

Format:	U6-1

The maximum value of the thread width counter. The width is this value + 1.

5 — 31:0 — Thread Group ID Starting X

This is the initial value of the X component of the thread group. When X reaches the maximum value it rolls around to this value.

GPGPU_WALKER		
6	31:0	**Reserved** Format: MBZ
7	31:0	**Thread Group ID X Dimension** The X dimension of the thread group (maximum X is dimension -1)
8	31:0	**Thread Group ID Starting Y** This is the initial value of the Y component of the thread group. When Y reaches the maximum value it rolls around to this value.
9	31:0	**Reserved** Format: MBZ
10	31:0	**Thread Group ID Y Dimension** The Y dimension of the thread group (maximum Y is dimension -1)
11	31:0	**Thread Group ID Starting/Resume Z** This is the initial value of the Z component of the thread group.
12	31:0	**Thread Group ID Z Dimension** The Z dimension of the thread group (maximum Z is dimension -1)
13	31:0	**Right Execution Mask** The execution mask used for threads on the right (maximum X) of the thread group.
14	31:0	**Bottom Execution Mask** The execution mask used for threads on the bottom (maximum Y) of the thread group.

Half Precision HI8DS Render Target Write MSD

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 3h Format: Opcode SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision LO8DS Render Target Write MSD

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 2h Format: Opcode SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision REP16 Render Target Write MSD

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision SIMD8 Render Target Write MSD

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 4h Format: Opcode SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision SIMD16 Render Target Write MSD

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 0h Format: Opcode SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Halt

halt - Halt

Source: Eulsa

Length Bias: 4

Description

The halt instruction temporarily suspends execution for all enabled compute channels. Upon execution, the enabled channels are sent to the instruction at (IP + UIP), if all channels are enabled at HALT, jump to the instruction at (IP + JIP).

If the halt instruction is not inside any conditional code block, the values of JIP and UIP should be the same. If the halt instruction is inside a conditional code block, the UIP should be the end of the program and the JIP should be the end of the inner most conditional code block.

The UIP must point to a HALT Instruction.

If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate and dst must be null.

Format:

[(pred)] halt (exec_size) JIP UIP

Syntax

```
[(pred)] halt (exec_size) imm32 imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn.channel[n] ) { PcIP[n] =
IP + UIP; else { PcIP[n] = IP + 1; } } if ( PcIP != (IP + 1) ) { // for all
channels Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS

halt - Halt			
	31:0	**Header**	
		Format:	EU_INSTRUCTION_HEADER

HCP_BSD_OBJECT

HCP_BSD_OBJECT

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_BSD_OBJECT command fetches the HEVC bit stream for a slice starting with the first byte in the slice. The bit stream ends with the last non-zero bit of the frame and does not include any zero-padding at the end of the bit stream. There can be multiple slices in a HEVC frame and thus this command can be issued multiple times per frame.

The HCP_BSD_OBJECT command must be the last command issued in the sequence of batch commands before the HCP starts decoding. Prior to issuing this command, it is assumed that all configuration parameters in the HCP have been loaded including workload configuration registers and configuration tables. When this command is issued, the HCP is waiting for bit stream data to be presented to the shift register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 20h HCP_BSD_OBJECT_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 1h — Name:

HCP_BSD_OBJECT

1	31:0	**Indirect BSD Data Length** Format: U32 Specifies the length in bytes of the bitstream data for the current slice. It includes the first byte of the slice and the last non-zero byte of the in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. **Value** / **Name** / **Description** [0,268435455] / Data_Length_with_28_bits_only / Valid range is only from 0 to 268435455, which is corresponding to lower 28 bits.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect Data Start Address** Format: U29 Specifies the byte-aligned graphics memory starting address of the slice bit stream relative to the **BSD Indirect Object Base Address**.

HCP_FQM_STATE

HCP_FQM_STATE

Source: VideoCS

Length Bias: 2

The HCP_FQM_STATE command loads the custom HEVC quantization tables into local RAM and may be issued up to 8 times: 4 scaling list per intra and inter.

Driver is responsible for performing the Scaling List division. So, save the division HW cost in HW. The 1/x value is provided in 16-bit fixed-point precision as ((1«17)/QM +1) » 1. .

Note: FQM is computed as (2^16)/QM. If QM=1, FQM=all 1's.

To simplify the design, only a limited number of scaling lists are provided at the PAK interface: default two SizeID0 and two SizeID123 (one set for inter and the other set for intra), and the encoder only allows custom entries for these four matrices. The DC value of SizeID2 and SizeID3 will be provided.

When the scaling_list_enable_flag is set to disable, the scaling matrix is still sent to the PAK, and with all entries programmed to the same value of 16.

This is a picture level state command and is issued in encoding processes only.

Dwords 2-33 form a table for the DCT coefficients, 2 16-bit coefficients/DWord.

- Size 4x4 for SizeID0, DWords 2-9.
- Size 8x8 for SizeID1/2/3, DWords 2-33.

SizeID 0 (Table 4-13)

4x4	[31:16]	[15:0]
DWord 2	AC(0,1)	DC
DWord 3	AC(0,3)	AC(0,2)
DWord 4	AC(1,1)	AC(1,0)
DWord 5	AC(1,3)	AC(1,2)
DWord 6	AC(2,1)	AC(2,0)
DWord 7	AC(2,3)	AC(2,2)
DWord 8	AC(3,1)	AC(3,0)
DWord 9	AC(3,3)	AC(3,2)

SizeID 1, 2, 3 (Table 4-14)

8x8	[31:16]	[15:0]
DWord 2	AC(0,1)	DC
DWord 3	AC(0,3)	AC(0,2)
DWord 4	AC(0,5)	AC(0,4)
DWord 5	AC(0,7)	AC(0,6)
DWord 6	AC(1,1)	AC(1,0)

HCP_FQM_STATE

DWord 7	AC(1,3)	AC(1,2)
DWord 8	AC(1,5)	AC(1,4)
DWord 9	AC(1,7)	AC(1,6)
...		
DWord 30	AC(7,1)	AC(7,0)
DWord 31	AC(7,3)	AC(7,2)
DWord 32	AC(7,5)	AC(7,4)
DWord 33	AC(7,7)	AC(7,6)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 5h HCP_FQM_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 20h Name:
1	31:16	**FQM DC Value: (1/DC):** Format: U16 Specifies DC value of the scaling list for 16x16 (SizeID=2) or 32x32 (SizeID=3). DC Value = scaling_list_dc_coef_minus8 + 8. Driver will do the division.

HCP_FQM_STATE

	15:5	**Reserved** Format: U11
	4:3	**Color Component** Format: U2 Luma and Chroma's share the same scaling list and DC value for the same SizeID. 0 – Luma 1 – Chroma Cb 2 – Chroma Cr 3 – Reserved
	2:1	**SizeID** Format: U2 0 – SizeID 0 4x4 1 – SizeID 1, 2, 3 (8x8, 16x16, 32x32) 2 – SizeID 2 (for DC value in 16x16) 3 – SizeID 3 (for DC value in 32x32)
	0	**Intra/Inter** Format: U1 This field specifies the quant matrix intra or inter type. 0 – Intra 1 – Inter
2..33	1023:0	**See Tables 4-13 and 4-14**

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_IND_OBJ_BASE_ADDR_STATE command is used to define the indirect object base address of the stream in graphics memory. This is a frame level command. (Is it frame or picture level?)
This is a picture level state command and is issued in both encoding and decoding processes.

Compressed Header Format

Fields	Bits	
Bin	0	Kernel Binarized Syntax
Probability select	1	0 ->indicates probability 128 1 ->indicates probability 256
	Repeat to pack a Cacheline	

Partition1 and TileSize record

Fields	Bits	
Tile Size	31:0	Partition1 Size is 16-bit value, Tile Size is 32-bit value
AddressOffset	63:32	Cacheline Address Offset to be Modified
Offset	69:64	Byte offset to be Modified
16-bit vs 32-bit update	70	0: Update 16-bit; 1: Update 32-bit
Reserved	511:71	

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 3h HCP_IND_OBJ_BASE_ADDR_STATE Format: OpCode

HCP_IND_OBJ_BASE_ADDR_STATE

	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: Ch — Name:
1..2	63:0	**HCP Indirect Bitstream Object Base Address** Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the HCP_BSD_OBJECT command for fetching (reading) the compressed Slice Data.
3	31:0	**HCP Indirect Bitstream Object Memory Address Attributes** Format: MemoryAddressAttributes
4..5	63:0	**HCP Indirect Bitstream Object Access Upper Bound** Format: SplitBaseAddress4KByteAligned Decoder only. This field specifies the 4K-byte aligned maximum memory address access by the indirect data object in the HCP_BSD_OBJECT command for the slice bit stream. Indirect data accessed at this address or greater will cause the HCP to stop issuing requests to the GAC and the BSP VLD will then only receive zeros until a slice done is received. Setting this field to 0 will cause this range to be ignored by the HCP.
6..7	63:0	**HCP Indirect CU Object Base Address** Format: BaseAddress4KByteAligned Encoder only. Specifies the 4K-byte aligned data buffer base address for the read-only indirect data object for fetching (reading) per CU data during the encoding process.
8	31:0	**HCP Indirect CU Object Object Memory Address Attributes** Format: MemoryAddressAttributes
9..10	63:0	**HCP PAK-BSE Object Base Address** Format: BaseAddress4KByteAligned Encoder only. Specifies the 4K-byte aligned memory base address for the write-only data pointed by the PAK engine for writing out the compressed bitstream.

HCP_IND_OBJ_BASE_ADDR_STATE		
11	31:0	**HCP PAK-BSE Object Address Memory Address Attributes** Format: **MemoryAddressAttributes** Encoder only.
12..13	63:0	**HCP PAK-BSE Object Access Upper Bound** Format: **SplitBaseAddress4KByteAligned** Encoder only. This field specifies the 4K-byte aligned maximum memory address access by the HCP_PAK_OBJECT command for writing out the slice bit stream. Indirect data accessed at this address and beyond will be blocked by the hardware and ignored. This address must be greater than the HCP PAK-BSE Object Base Address state.

HCP_PAK_INSERT_OBJECT

HCP_PAK_INSERT_OBJECT	
Source:	VideoCS
Length Bias:	2

It is an encoder only command, operating at bitstream level, before and after SliceData compressed bitstream. It is setup by the header and tail present flags in the Slice State command. If these flags are set and no subsequent PAK_INSERT_OBJECT commands are issued, the pipeline will hang.
The HCP_ PAK_ INSERT _OBJECT command supports both inline and indirect data payload, but only one can be active at any time. It is issued to insert a chunk of bits (payload) into the current compressed bitstream output buffer (specified in the HCP_PAK-BSE Object Base Address field of the HCP_IND_OBJ_BASE_ADDR_STATE command) starting at its current write pointer bit position. Hardware will keep track of this write pointer's byte position and the associated next bit insertion position index.
It is a variable length command when the payload (data to be inserted) is presented as inline data within the command itself. The inline payload is a multiple of 32-bit (1 DW), as the data bus to the compressed bitstream output buffer is 32-bit wide.
The payload data is required to be byte aligned on the left (first transmitted bit order) and may or may not be byte aligned on the right (last transmitted bits). The command will specify the bit offset of the last valid DW. Note that : Stitch Command is used if the beginning position of data is in bit position. When PAK Insert Command is used the beginning position must be in byte position.
Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid bitstream.
Internally, HCP hardware will keep track of the very last two bytes' (the very last byte can be a partial byte) values of the previous insertion. It is required that the next Insertion Object Command or the next PAK Object Command to perform the start code emulation sequence check and prevention 0x03 byte insertion with this end condition of the previous insertion.
The payload data may have already been processed for start code emulation byte insertion, except the possibility of the last 2 bytes plus the very last partial byte (if any). Hence, when hardware performing the concatenation of multiple consecutive insertion commands, or concatenation of an insertion command and a PAK object command, it must check and perform the necessary start code emulation byte insert at the junction.
Data to be inserted can be a valid NAL units or a partial NAL unit. It can be any encoded syntax elements bit data before the encoded Slice Data (PAK Object Command) of the current Slice - SPS NAL, PPS NAL, SEI NAL and Other Non-Slice NAL, Leading_Zero_8_bits (as many bytes as there is), Start Code , Slice Header. Any encoded syntax elements bit data after the encoded Slice Data (PAK Object Command) of the current Slice and prior to the next encoded Slice Data of the next Slice or prior to the end of the bitstream, whichever comes first Cabac_Zero_Word or Trailing_Zero_8bits (as many bytes as there is).
Certain NAL unit has a minimum byte size requirement. As such the hardware will optionally (enabled by SLICE STATE Command) determines the number of CABAC_ZERO_WORD to be inserted to the end of the current NAL, based on the minimum byte size of a NAL and the actual bin count of the encoded Slice. Since prior to the CABAC_ZERO_WORD insertion, the RBSP or EBSP is already byte-aligned, so each CABAC_ZERO_WORD insertion is actually a 3-byte sequence 0x00 00 03.
Context switch interrupt is not supported by this command.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="6">0</td><td>31:29</td><td>Command Type<table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>28:27</td><td>Pipeline Type<table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>26:23</td><td>Media Instruction Opcode<table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HCP = 7h</td></tr>
<tr><td>22:16</td><td>Media Instruction Command<table><tr><td>Default Value:</td><td>22h HCP_PAK_INSERT_OBJECT</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>15:12</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>11:0</td><td>Dword Length<table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1) = N-1, Variable Length in DW if inline data; =4, fixed length if indirect data.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[1,4095]</td><td></td></tr></table></td></tr>
<tr><td rowspan="4">1</td><td>31</td><td>Indirect Payload Enable<table><tr><td>Format:</td><td>Enable</td></tr></table>Only one of these two payload modes can be active at any time.
When Slice Size Conformance is enable the Payload(header) must be inline only so this bit set to MBZ.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>inline payload is used</td><td></td></tr><tr><td>1</td><td>indirect payload is used</td><td></td></tr></table></td></tr>
<tr><td>30:18</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>17:16</td><td>DataByteOffset - SrcDataStartingByteOffset[1:0]<table><tr><td>Format:</td><td>U2</td></tr></table>Source Data Starting Byte Position within the very first inline DW.</td></tr>
</table>

HCP_PAK_INSERT_OBJECT

15 **HeaderLengthExcludeFrmSize**

Format:	U1

In case this flag is on, bits are NOT accumulated during current access unit coding neither for Cabac Zero Word insertion bits counting or for output in MMIO register HCP_BITSTREAM_BYTECOUNT_FRAME_NO_HEADER.
When using HeaderLenghtExcludeFrmSize for header insertion, the software needs to make sure that data comes already with inserted start code emulation bytes. SW shouldn't set EmulationFlag bit (Bit 3 of DWORD1 of HCP_PAK_INSERT_OBJECT).

Value	Description
0	All bits accumulated
1	Bits during current call are not accumulated

Value	Name	Description
0	All bits accumulated	
1	Bits during current call are not accumulated	

Programming Notes
Must be set to 0 for JPEG encoder.

13:8 **DataBitsInLastDW - SrCDataEndingBitInclusion[5:0]**

Format:	U6

Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first.For example, SrCDataEndingBitInclusion = 9, bit 7:0 and bit 15 are included as valid header data.
For VP9:
The Driver has to give byte aligned Data for the last inline DW. (the driver pads Zeros to next byte boundary to the original header if it was not byte aligned on the last inline DW).

Value	Name
[1,32]	

7:4 **SkipEmulByteCnt - Skip Emulation Byte Count**

Format:	U4

Skip emulation check for number of starting bytes It can be programmed from 0 to 15 bytes. For example, to skip the start code that has already prefixed in the bitstream.
Only valid for HEVC and reserved for VP9.

3 **EmulationFlag - EmulationByteBitsInsertEnable**

Format:	U1

Only valid for HEVC and reserved for VP9.

Value	Name	Description
1	StartCodePrefix	Instruct the hardware to perform Start Code Prefix (0x 00 00 01/02/03/00) Search and Prevention Byte (0x 03) insertion on the

HCP_PAK_INSERT_OBJECT

				insertion data of this command. It is required that hardware will handle a start code prefix crossing the boundary between.
		2	InsertionCommand	Insertion commands, or an insertion command followed by a PAK Object command.
	2	**LastHeaderFlag - LastSrcHeaderDataInsertCommandFlag** Format: U1 To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series. In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.
	1	**EndOfSliceFlag - LastDstDataInsertCommandFlag** Format: U1 No more insertion command and no more PAK-OBJECT command follows. Flush data out to memory.
	0	**Reserved** Format: MBZ
2..n	127:0	**Indirect Payload** Exists If: ([Indirect Payload Enable]==1) Format: **HCP_PAK_INSERT_OBJECT_INDIRECT_PAYLOAD**
	31:0	**Inline PayLoad** Exists If: ([Indirect Payload Enable]==0) Format: U32 Actual Data (inline) to be inserted to the output bitstream buffer.

HCP_PAK_OBJECT

HCP_PAK_OBJECT

Source: VideoCS

Length Bias: 2

This is a per-LCU command for encoder only.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 21h HCP_PAK_OBJECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 1h — Name: Value: 3h — Name:
1	31	**LastCtbOfSlice Flag** Format: Boolean Value: 0 — Name: False Value: 1 — Name: True

HCP_PAK_OBJECT		
	30	**LastCtbOfTile Flag** Format: U1 Indicates last LCU or SB of a Tile
	29:24	**CU_count_minus1** Format: U6 Number of CUs in the current LCU = CU_count_minus1 + 1. Minimum, there must be 1 CU in a LCU.
	23:21	**Reserved** Format: MBZ
	20:0	**split_coding_unit_flag[x0][y0]** Format: Split_coding_unit_flags If CU size=64x64 and DQP!=0, split should happen at least once
2	31:16	**Current LCU Y Addr** Format: U16
	15:0	**Current LCU X Addr** Format: U16
3..4	31:0	**Reserved** Format: MBZ

HCP_PIC_STATE

HCP_PIC_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a picture level command and is issued only once per workload for both encoding and decoding processes.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 10h HCP_PIC_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value / Name: 11h; 1Dh
1	31:26	**Reserved** Format: MBZ
	25:16	**FrameHeightInMinCbMinus1** Format: U10 Specifies the height of each decoded picture in units of minimum coding block size.

HCP_PIC_STATE

		Value / **Name**: [0-4122] /
		Programming Notes: The decoded picture height in units of luma samples equals • (FrameHeightInMinCbMinus1 + 1) * • (1 « (log2_min_coding_block_size_minus3 + 3))
		In HEVC Encoder mode, the following restrictions apply. Note : for a frame height that is not an integer multiple of LCU : Kernel : on last LCU at frame's bottom edge, it must align to CU boundary. This applies to all size of LCU: 16x16, 32x32, 64x64. Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively. Driver : sets up a LCU aligned (both in X/Y direction) surface.
	15	**PAK Transform Skip Enable** If global bit, **transform_skip_enabled_flag** in image state dw4.22 set to 1 and this bit set to, 1 -> HW will perform transform skip and over write the ENC decision in CU record 0-> HW will not perform transform skip if **transform_skip_enabled_flag=0**, no transform skip performed in HW or ENC **Programming Notes**: Encoder Only
	14:10	**Reserved** Format: MBZ
	9:0	**FrameWidthInMinCbMinus1** Format: U10 Specifies the width of each decoded picture in units of minimum coding block size. **Value** / **Name**: [0-4122] / **Programming Notes**: The decoded picture width in units of luma samples equals • (FrameWidthInMinCbMinus1 + 1) * • (1 « (log2_min_coding_block_size_minus3 + 3))
2	15	**Reserved** Format: MBZ
	31:12	**Reserved** Format: MBZ

HCP_PIC_STATE

11:10 MaxPCMSize

Format:	U2

Specifies the largest allowed PCM coding block size.

Value	Name
3	Reserved
2	32x32
1	16x16
0	8x8

9:8 MinPCMSize

Format:	U2

Specifies the smallest allowed PCM coding block size.

Value	Name
3	Reserved
2	32x32
1	16x16
0	8x8

7:6 MaxTUSize

Format:	U2

Specifies the largest allowed transform block size.

Value	Name
3	32x32
2	16x16
1	8x8
0	4x4

5:4 MinTUSize

Format:	U2

Specifies the smallest allowed transform block size.

Value	Name
3	32x32
2	16x16
1	8x8
0	4x4

3:2 CtbSize (LCUSize)

Format:	U2

Specifies the coding tree block size.

Value	Name

HCP_PIC_STATE

		3 64x64 2 32x32 1 16x16 0 illegal/reserved **Programming Notes** LCU is restricted based on the picture size.
	1:0	**MinCUSize** Format: U2 Specifies the smallest coding block size. 3 64x64 2 32x32 1 16x16 0 8x8
3	31:3	**Reserved** Format: MBZ

	2	**InsertTestFlag**
		Format: U1
		
		0h **[Default]**
		1h
		
		Programming Notes
		CABAC 0 Word Insertion Test Enable (Encoder Only)This bit will modify CABAC K equation so that a positive K value can be generated easily. This is done for validation purpose only. In normal usage this bit should be set to 0.
		Regular equation for generating 'K' value when CABAC 0 Word Insertion Test Enable is set to 0.
		`K = {[((96 * pic_bin_count()) - (RawMinCUBits * PicSizeInMinCUs *3) + 1023) / 1024] - bytes_in_picture} / 3`
		Modified equation when CABAC 0 Word Insertion Test Enable bit set to 1.
		`K = {[((1536 * pic_bin_count()) - (RawMinCUBits * PicSizeInMinCUs *3) + 1023) / 1024] - bytes_in_picture} / 3`
		Encoder only feature.
		This bit should be set to 0 in regular PAK mode.

HCP_PIC_STATE

	1	**CurPicIsI** Format: U1 Specifies that the current picture is comprised solely of I slices and that there are no P or B slices in the picture. Value / Name: 0 – Current picture has at least one P or B slice **Programming Notes** This bit should be set to "0". Note: The value of "1" setting ("Current picture is comprised solely of I slices") is REMOVED. This bit is used for hardware optimization only. There is not enough information to set this bit to "1" correctly.
	0	**ColPicIsI** Format: U1 Specifies that the collocated picture is comprised solely of I slices and that there are no P or B slices in the picture. Value / Name: 0 – Collocated picture has at least one P or B slice **Programming Notes** This bit should be set to "0". Note: The value of "1" setting ("Collocated picture is comprised solely of I slices") is REMOVED. This bit is used for hardware optimization only. There is not enough information to set this bit to "1" correctly.
4	31:28	**Reserved** Format: MBZ
	27	**CU packet structure** Description Value / Name / Description: 0 / VME / CU packet structure is in output format of HW VME which is 1/2 CL per CU 1 / ExtEnc / CU packet structure is in output format of non HW VME (External Enc/SW) which is 1 CL per CU
	26	**strong_intra_smoothing_enable_flag** Format: U1

HCP_PIC_STATE

25 **transquant_bypass_enable_flag**

Format:	Enable

Value	Name	Description
0	Disable	cu_transquant_bypass is not supported
1	Enable	cu_transquant_bypass is supported

24 **Reserved**

Format:	MBZ

23 **amp_enabled_flag**

Format:	Enable

Value	Name	Description
0	Disable	Asymmetric motion partitions cannot be used in coding tree blocks.
1	Enable	Support asymmetric motion partitions, i.e. PartMode equal to PART_2NxnU, PART_2NxnD, PART_nLx2N, or PART_nRx2N.

22 **transform_skip_enabled_flag**

Format:	Enable

Value	Name	Description
0	Disable	transform_skip_flag is not supported in the residual coding
1	Enable	transform_skip_flag is supported

21 **BottomField**

Format:	U1

Value	Name	Description
0	Bottom Field	
1	Top Field	

Programming Notes
Must be zero for encoder only

20 **FieldPic**

Format:	U1

Value	Name	Description
0	Video Frame	
1	Video Field	

Programming Notes

HCP_PIC_STATE

		Must be zero for encoder only.
	19	**weighted_pred_flag** Format: U1
	18	**weighted_bipred_flag** Format: U1
	17	**tiles_enabled_flag** Format: U1 **Programming Notes** Tiling is not supported and this bit should be set to 0.
	16	**entropy_coding_sync_enabled_flag** Format: U1 Not used in encoder mode
	15	**Reserved**
	15	**loop_filter_across_tiles_enabled_flag** Format: U1
	14	**Reserved** Format: MBZ
	13	**sign_data_hiding_flag** Format: Enable **Programming Notes** Currently not supported in encoder, so must be set to 0 for encoding session.
	12:10	**log2_parallel_merge_level_minus2** Format: U3 **Programming Notes** For encoder, always set to 0 (Intel restriction).

Value	Name	Description
0	Disable	Specifies that sign bit hiding is disabled.
1	Enable	Specifies that sign bit hiding is enabled.

Value	Name	Programming Notes
[0,4]	Valid Range	The value of log2_parallel_merge_level_minus2 shall be in the range of 0 to Log2CtbSizeYCtbLog2SizeY - 2, inclusive.

HCP_PIC_STATE

	9	**constrained_intra_pred_flag** Format: U1
	8	**pcm_loop_filter_disable_flag** Format: U1
	7:6	**diff_cu_qp_delta_depth (or named as max_dqp_depth)** Format: U2 **Programming Notes** cu_qp_delta_enabled_flag = 1 and Max_DQP_Level = 0 or 3is supported in PAK standalone mode. cu_qp_delta_enabled_flag/max_dqp_depth: 1/0: has cu qp delta. (cu depth <= max_dqp_depth) will have cu qp delta coded. Only allow QP change across Slices, no change across LCU or CU.
	5	**cu_qp_delta_enabled_flag** Format: U1 **Value** / **Name** / **Description** 0 / Disable / Does not allow QP change at CU or LCU level, the same QP is used for the entire slice. Max_DQP_Level = 0 (i.e. diff_cu_qp_delta_depath = 0). 1 / Enable / Allow QP change at CU level. MAX_DQP_level can be >0. **Programming Notes** cu_qp_delta_enabled_flag = 1 and Max_DQP_Level = 0 or 3 is supported for PAK standalone mode.
	4	**pcm_enabled_flag** Format: U1
	3	**sample_adaptive_offset_enabled_flag** Format: U1 **Programming Notes** HW does not support SAO filtering in encoder mode
	2:0	**Reserved** Format: MBZ
5	31:30	**Reserved** Format: MBZ

HCP_PIC_STATE

29:27 **bit_depth_luma_minus8**

Format:	U3

This specifies the number of bit allow for Luma pixels. In 8 bit mode, this must be set to 0. Encoder: Suports bit depths 8, 10 and 12 only. Encoder: Does not support 10 or 12 bit Source Pixels and 8bit PAK i.e. the source pixel depth should be less than or equal to PAK bit depth.

Value	Name	Description
0	luma_8bit	
1	luma_9bit	Only HEVC decoder supports 9 bits luma. HEVC encoder does not supports 9 bits luma.
2	luma_10bit	
3	luma_11bit	
4	luma_12bit	

26:24 **bit_depth_chroma_minus8**

Format:	U3

This specifies the number of bit allow for Chroma pixels. In 8 bit mode, this must be set to 0. Encoder: Supports bit depths 8, 10 and 12 only. And also it must be same as Luma. Encoder: Does not support 10 or 12 bit Source Pixels and 8bit PAK. i.e. The source pixel depth should be less than or equal to the PAK bit depth.

Value	Name	Description
0	chroma_8bit	
1	chroma_9bit	Only HEVC decoder supports 9 bits chroma. HEVC encoder does not supports 9 bits chroma.
2	chroma_10bit	
3	chroma_11bit	
4	chroma_12bit	

23:20 **pcm_sample_bit_depth_luma_minus1**

Format:	U4

Description
Encoder support bitdepth 8 only. Must be same for luma and chroma Cb and Cr.

19:16 **pcm_sample_bit_depth_chroma_minus1**

Format:	U4

Description
Encoder support bitdepth 8 only. Must be same for luma and chroma Cb and Cr.

HCP_PIC_STATE

	15:13	**max_transform_hierarchy_depth_inter(or named as tu_max_depth_inter)** Format: U3 Maximum TU split depths for inter blocks **Programming Notes** For encoder, alwasy set to 2 to allow max 2 levels of split. For more splitting rely on CU split to match the content (Intel restriction)
	12:10	**max_transform_hierarchy_depth_intra (or named as tu_max_depth_intra)** Format: U3 Maximum TU split depth for intra blocks. **Programming Notes** For encoder, always set to 2 to allow max 2 levels of split. For more splitting, rely on CU split to match the content (Intel restriction).
	9:5	**pic_cr_qp_offset** Format: S4 Valid range -12 to +12
	4:0	**pic_cb_qp_offset** Format: S4 Valid range -12 to +12
6	31:30	**Reserved** Format: MBZ
	29	**Load Slice Pointer Flag** Format: Enable LoadBitStreamPointerPerSlice (Encoder-only) To support multiple slice picture and additional header/data insertion before and after an encoded slice. When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream. When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0	Disable	Load BitStream Pointer only once for the first slice of a frame.
1	Enable	Load/reload BitStream Pointer only once for the each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field.

Programming Notes

Must be zero for encoder

HCP_PIC_STATE

28:27 **Reserved**

Format:	MBZ

26 **FrameSzUnderStatusEn - FrameBitRateMinReportMask**

Format:	Enable

This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin.

Value	Name	Description
0	Disable	Do not update bit 2 (Frame Bit Count Violate -- under run) of HCP_IMAGE_STATUS control register. NOTE: This bit MUST BE set to zero for the last BRC pass if SAO first pass also enabled. This will ensure that HW picks up right accumulated deltaQP for SAO sencond pass.
1	Enable	Set bit 2 (Frame Bit Count Violate -- under run) of HCP_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit Rate Minimum limit. HW does not use this bit to set the bit in HCP_IMAGE_STATUS_CONTROL register. It's used pass the bit in HCP_IMAGE_STATUS_MASK register

25 **FrameSzOverStatusEn - FrameBitRateMaxReportMask**

Format:	Enable

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0	Disable	Do not update bit 1 of HCP_IMAGE_STATUS control register. NOTE: This bit MUST BE set to zero for the last BRC pass if SAO first pass also enabled. This will ensure that HW picks up right accumulated deltaQP for SAO sencond pass.
1	Enable	Set bit 1 of HCP_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit Rate Maximum limit. HW does not use this bit to set the bit in HCP_IMAGE_STATUS_CONTROL register. It's used pass the bit inHCP_IMAGE_STATUS_MASK register

24 **LCUMaxBitStatusEn - LCUMaxSizeReportMask**

Format:	Enable

This is a mask bit controlling if the condition of any LCU in the frame exceeds LCUMaxSize.

Value	Name	Description
0	Disable	Do not update bit 0 of HCP_IMAGE_STATUS control register. NOTE: This bit MUST BE set to zero for the last BRC pass if SAO first pass also enabled. This will ensure that HW picks up right accumulated

HCP_PIC_STATE

		deltaQP for SAO sencond pass.
1	Enable	Set bit 0 of HCP_IMAGE_STATUS control register if the total bit counter for the current LCU is greater than the LCU Conformance Max size limit. HW does not use this bit to set the bit in HCP_IMAGE_STATUS_CONTROL register.

Programming Notes
Encoder Only

23:17 **Reserved**

Format:	MBZ

16 **NonFirstPassFlag**

Format:	Enable

This signals the current pass is not the first pass. It will imply designate HW behavior.

Value	Name	Description
0	Disable	If it is initial-Pass, this bit is set to 0.
1	Enable	For subsequent passes, this bit is set to 1.

Programming Notes
Encoder Only

15:0 **LCU Max BitSize Allowed**

Format:	U16

This field specifies the Max LCU Bit Size allowed in according to the spec conformance. However, driver can program a different value from the spec for other encoding purpose.

If LCU Limit exceeds, LCUMaxBitStatus in MMIO would be set to 1.
If LCU Limit exceeds, LCUMaxBitStatus in MMIO would be set to 1 and also in CU. Statistics Streamout, LCU Limit exceed field would be set to 1 at the last CU of a LCU.

Programming Notes
Encoder Only

7 **Programming Notes:** Encoder Only

31 **FrameBitrateMaxUnit**

Format:	U1

This field is the Frame Bitrate Maximum Limit Units.

Value	Name	Description
0	Byte	32byte unit
1	Kilo Byte	4kbyte unit

HCP_PIC_STATE

		Programming Notes Encoder Only
	30:14	**Reserved** Format: MBZ Reserved for future expansion of the Max Rate.
	13:0	**FrameBitRateMax** Format: U14 This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. **Value** / **Name** / **Description** 0-512KB / / The programmable range is 0-512KB when FrameBitrateMaxUnit is 0. 0-64MB / / The programmable range is 0-64Mbyte when FrameBitrateMaxUnit is 1.
8	31	**FrameBitrateMinUnit** Format: U1 This field is the Frame Bitrate Minimum Limit Units. **Value** / **Name** / **Description** 0 / Byte / 32byte unit 1 / Kilo Byte / 4kbyte unit **Programming Notes** Encoder Only
	30:14	**Reserved** Format: MBZ Reserved for future expansion of the Min Rate.
	13:0	**FrameBitRateMin** Format: U14 This field is the Frame Bitrate Minimum Limit. This field along with FrameBitrateMinUnit determines minimum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count happen to be below this value. **Value** / **Name** / **Description** 0-512KB / / The programmable range is 0-512KB when FrameBitrateMinUnit is 0.

HCP_PIC_STATE

		0-64MB | | The programmable range is 0-64Mbyte when FrameBitrateMinUnit is 1.
		Programming Notes: Encoder Only
9	31	**Reserved** Format: MBZ
	30:16	**FrameBitRateMaxDelta** Exists If: //Always Format: U15 This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit.

Value	Name	Description
0	**[Default]**	
0-1024KB		The Programmable range is 0-1024KB when FrameBitRateMaxUnit is 0.
0-128MB		The Programmable range is 0-128MB when FrameBitRateMaxUnit is 1.

Programming Notes: Encoder Only

	15	**Reserved** Format: MBZ
	14:0	**FrameBitRateMinDelta** Format: U15 This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit.

Value	Name	Description
0	**[Default]**	
0-1024KB		The Programmable range is 0-1024KB when FrameBitRateMinUnit is 0.
0-128MB		The Programmable range is 0-128MB when FrameBitRateMinUnit is 1.

Programming Notes: HW requires the following condition: FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior.

Programming Notes: Encoder Only

HCP_PIC_STATE

10..11	63:0	**FrameDeltaQpMax** Format: FrameDeltaQp Range: [0:MAX_QP_DELTA] Frame level delta QP which should be used in case FrameSize - FrameBitRateMax in the range of ((DeltaQpMaxRange[n] * FrameBitRateMaxDelta»5)), DeltaQpMaxRange[n+1] * FrameBitRateMaxDelta»5)). **Programming Notes** If n == 7, DeltaQpMaxRange is infinity. Format: 8 bit sign-magnitude, Range: [0:63] Encoder Only
12..13	63:0	**FrameDeltaQpMin** Format: FrameDeltaQp Range: [0:MIN_QP_DELTA] Frame level delta QP which should be used in case FrameSize - FrameBitRateMin in the range of ((DeltaQpMinRange[n] * FrameBitRateMinDelta»5)), DeltaQpMinRange[n+1] * FrameBitRateMinDelta»5)). **Programming Notes** If n == 7, DeltaQpMinRange is infinity. Format: 8 bit sign-magnitude Range: [-63:0] Encoder Only
14..15	63:0	**FrameDeltaQpMaxRange** Format: FrameDeltaQpRange Range: [0:U8_MAX] Condition: FrameDeltaQpMaxRange[n] >= FrameDeltaQpMaxRange[n-1] This field is to calculate ranges for Frame level delta QP, specifically Frame level delta QP[n] and Frame level delta QP[n+1]. **Programming Notes** If n == 0, FrameDeltaQpMaxRange is zero. Encoder Only
16..17	63:0	**FrameDeltaQpMinRange** Format: FrameDeltaQpRange Range: [0:U8_MAX]

HCP_PIC_STATE

		Condition: FrameDeltaQpMinRange[n] >= FrameDeltaQpMinRange[n-1] This field is to calculate ranges for Frame level delta QP, specifically Frame level delta QP[n] and Frame level delta QP[n+1]. **Programming Notes** If n == 0, FrameDeltaQpMinRange is zero. Encoder Only
18	31:30	**MinFrameSizeUnits** Format: U2 This field is the Minimum Frame Size Units **Value** / **Name** / **Description** 0 / 4Kb / Minimum Frame Size is in 4Kbytes. 1 / 16Kb / Minimum Frame Size is in 16Kbytes. 2 / Reserved / 3 / Reserved / **Programming Notes** Encoder Only
	29:16	**Reserved** Format: MBZ
	15:0	**MinFrameSize** Default Value: 0 Format: U16 Minimum Frame Size [15:0] (in Word, 16-bit)(Encoder Only) Mininum Frame Size is specified to compensate for intel Rate Control Currently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. It is needed for CBR. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Mininum Frame Size, is always less than maximum frame size FrameBitRateMax. This field is reserved in Decode mode. **Programming Notes** Programmable range is 0..(2^16-1) * 2^12 when MinFrameSizeUnits is 0. (4KB unit) Programmable range is 0..(2^16-1) * 2^14 when MinFrameSizeUnits is 1. (16KB unit) Encoder Only
19..31	31:0	**Reserved** Format: MBZ

HCP_PIPE_BUF_ADDR_STATE

HCP_PIPE_BUF_ADDR_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This state command provides the memory base addresses for the row store buffer and reconstructed picture output buffers required by the HCP.
This is a picture level state command and is shared by both encoding and decoding processes.

Programming Notes

All pixel surface addresses must be 4K byte aligned. There is a max of 8 Reference Picture Buffer Addresses, and all share the same third address DW in specifying 48-bit address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 2h HCP_PIPE_BUF_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 51h, 66h Name:

HCP_PIPE_BUF_ADDR_STATE		
1..2	63:0	**Decoded Picture** Format: SplitBaseAddress4KByteAligned Frame buffer address for the final decoded picture YUV output.
3	31:0	**Decoded Picture** Format: MemoryAddressAttributes
4..5	63:0	**Deblocking Filter Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
6	31:0	**Deblocking Filter Line Buffer** Format: MemoryAddressAttributes
7..8	63:0	**Deblocking Filter Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
9	31:0	**Deblocking Filter Tile Line Buffer** Format: MemoryAddressAttributes
10..11	63:0	**Deblocking Filter Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
12	31:0	**Deblocking Filter Tile Column Buffer** Format: MemoryAddressAttributes
13..14	63:0	**Metadata Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Line buffer.
15	31:0	**Metadata Line Buffer** Format: MemoryAddressAttributes
16..17	63:0	**Metadata Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Line buffer.
18	31:0	**Metadata Tile Line Buffer** Format: MemoryAddressAttributes

HCP_PIPE_BUF_ADDR_STATE

19..20	63:0	**Metadata Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Column buffer.
21	31:0	**Metadata Tile Column Buffer** Format: MemoryAddressAttributes
22..23	63:0	**SAO Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Line buffer.
24	31:0	**SAO Line Buffer** Format: MemoryAddressAttributes
25..26	63:0	**SAO Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Tile Line buffer.
27	31:0	**SAO Tile Line Buffer** Format: MemoryAddressAttributes
28..29	63:0	**SAO Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Tile Column buffer.
30	31:0	**SAO Tile Column Buffer** Format: MemoryAddressAttributes
31..32	63:0	**Current Motion Vector Temporal Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Current Motion Vector Temporal buffer.
33	31:0	**Current Motion Vector Temporal Buffer** Format: MemoryAddressAttributes
34..35	63:0	**Reserved** Format: MBZ
36	31:0	**Reserved** Format: MBZ
37..52	63:0	**Reference Picture Base Address (RefAddr[0-7])** Format: SplitBaseAddress64ByteAligned[8] Base address of the reference picture buffer.

HCP_PIPE_BUF_ADDR_STATE

53	31:0	**Reference Picture Base Address** Format: **MemoryAddressAttributes**
54..55	63:0	**Original Uncompressed Picture Source** Format: **SplitBaseAddress64ByteAligned** Buffer address for fetching YUV pixel data from the original uncompressed input picture for encoding. This value is only valid in **encoding** mode.
56	31:0	**Original Uncompressed Picture Source** Format: **MemoryAddressAttributes**
57..58	63:0	**Streamout Data Destination** Exists If: //Decoder Only Format: **SplitBaseAddress64ByteAligned** Buffer address for outputting the per-block indirect data to memory when **StreamOutEnable** is set in the HCP_PIPE_MODE_SELECT command. For Decoder: this field is used for transcoding purpose. For Encoder: this field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit and ENC Unit. For Encoder: This surface is used to streamout CU records All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-block data.
59	31:0	**Streamout Data Destination** Exists If: //Decoder Only Format: **MemoryAddressAttributes**
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
60..61	63:0	**Decoded Picture Status/Error Buffer Base Address or Encoded slice size streamout Base Address** Format: **SplitBaseAddress64ByteAligned** **Decoder Mode :** Specifies the 64 byte aligned buffer address for writing a single status/error cache-line sized record into memory when the Pic Status/Error Report Enable is set in the HCP_PIPE_MODE_SELECT command. The pic status/error record is written by hardware after the picture is decoded. **Encoder Mode:** This specifies 64 byte aligned buffer address for writing Slice size, when slice size conformance is enabled.

HCP_PIPE_BUF_ADDR_STATE

62	31:0	**Decoded Picture Status/Error Buffer Base Address** Format: **MemoryAddressAttributes**
63..64	63:0	**LCU ILDB Streamout Buffer** Format: **SplitBaseAddress64ByteAligned** Buffer address for writing ILDB parameter per LCU to memory when Deblocker Streamout Enable is set in the HCP_PIPE_MODE_SELECT Command. The ILDB MB control parameters are written by HW at the end of each reconstructed LCU. Only edge information is being streamed out.
65	31:0	**LCU ILDB Streamout Buffer** Format: **MemoryAddressAttributes**
66..81	63:0	**Collocated Motion Vector Temporal Buffer[0-7]** Format: **SplitBaseAddress64ByteAligned[8]** Base address for the Collocated Motion Vector Temporal buffer.
82	31:0	**Collocated Motion Vector Temporal Buffer[0-7] Memory Address Attributes** Format: **MemoryAddressAttributes**
83..84	63:0	**VP9 Probability Buffer Read/Write** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for VP9 Probability Buffer. Hardware reads in the probability for decode and write out the modified probability for future frames. Driver needs to program the Initial VP9 Probability for decoding the current frame. For Key Frame, it should contain the default Key Frame Probability. For non-Key Frame, it could be a default (non-Key) or one of the 8 Reference Buffers Probability. Driver must provide a valid Initial VP9 Probability buffer.
85	31:0	**VP9 Probability Buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
86..87	63:0	**VP9 Segment ID Buffer Read/Write** Specifies the 64 byte aligned buffer address for VP9 SegmentID buffer. This should contain the writeout SegmentID from previous frame and will be used to predict SegmentID for the current frame. Hardware will write out SegmentID of the current frame in the same address for the next frame.
88	31:0	**VP9 Segment ID buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
89..90	63:0	**VP9 HVD Line Rowstore Buffer Read/Write** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for HVD Tile Rowstore Buffer (bitstream decoder).

HCP_PIPE_BUF_ADDR_STATE		
91	31:0	**VP9 HVD Line Rowstore buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes
92..93	63:0	**VP9 HVD Tile Rowstore Buffer Read/Write** Format: SplitBaseAddress64ByteAligned
94	31:0	**VP9 HVD Tile Rowstore buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes
95..103	31:0	**Reserved** Format: MBZ

HCP_PIPE_MODE_SELECT

HCP_PIPE_MODE_SELECT

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.
The HCP_PIPE_MODE_SELECT command is responsible for general pipeline level configuration that would normally be set once for a single stream encode or decode and would not be modified on a frame workload basis.
This is a picture level state command and is shared by both encoding and decoding processes.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 0h HCP_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n (Excludes Dwords 0, 1). Value: 2h — Name: Value_2 Value: 4h — Name: Value_4
1	31:24	**Reserved**

HCP_PIPE_MODE_SELECT

	23	**Reserved** Format: MBZ
	22:20	**Reserved** Format: MBZ
	19:18	**Reserved** Format: MBZ
	17	**RESERVED** Format: MBZ
	16:13	**Reserved** Format: MBZ
	12	**PAK Frame Level StreamOut enable** Format: U1 This bit is valid if global bitPAK Pipeline Streamout Enableis set to 1. This bit is defined to use legacy tests on HW and it's valid for both hevc/vp9. By default it should be '0'
	11	**Reserved** Format: MBZ
	10	**Reserved**
	9	**Reserved**
	7:5	**Codec Standard Select** Value: 0 — Name: HEVC Value: 1 — Name: VP9
	4	**Reserved**
	3	**Pic Status/Error Report Enable** Format: Enable Value 0 — Name: Disable — Description: Disable status/error reporting Value 1 — Name: Enable — Description: Status/Error reporting is written out once per picture. The Pic Status/Error Report ID in DWord3 along with the status/error status bits are packed into one cache line and written to the Status/Error Buffer address in the HCP_PIPE_BUF_ADDR_STATE command. Must be zero for encoder mode.

HCP_PIPE_MODE_SELECT

	2	**PAK Pipeline Streamout Enable** Format: Enable Pipeline Streamout Enable is only defined for encode. It is ignored for decode. Value / Name: 0 – Disable pipeline states and parameters streamout 1 – Enable pipeline states and parameters streamout **Programming Notes** Shouldn't be turned off during SAO first PASS
	1	**Deblocker Streamout Enable** Format: Enable Deblocker Streamout Enable not currently supported for Encode or Decode Value / Name / Description: 0 – Disable – Disable deblocker-only parameter streamout 1 – Enable – Enable deblocker-only parameter streamout
	0	**Codec Select** Format: U1 Value / Name: 0 – Decode 1 – Encode
2	31:0	**Media Soft-Reset Counter (per 1000 clocks)** Format: U32 In decoder modes, this counter value specifies the number of clocks (per 1000) of GAC inactivity before a media soft-reset is applied to the HCP. If counter value is set to 0, the media soft-reset feature is disabled and no reset will occur. In encoder modes, this counter must be set to 0 to disable media soft reset. This feature is not supported for the encoder. Value / Name: 0 – Disable
3	31:0	**Pic Status/Error Report ID** Format: U32 The Pic Status/Error Report ID is a unique 32-bit unsigned integer assigned to each picture status/error output. Must be zero for encoder mode. Value / Name / Description: 0 – 32-bit unsigned – Unique ID Number 1 – Reserved

HCP_PIPE_MODE_SELECT

		Programming Notes Software must program different Status/Error Buffer addresses between pictures; otherwise the hardware might overwrite previously written data.
4	31:20	**Reserved**
	19	**Reserved**
	18	**Bitstream Decoder Error Handling Mode** **Programming Notes** This modifies how bitstream decoder handles error when there are extra bitstream within the tiles "0"--Decoder will trust the tile byte length and flush extra unused bitstream "1"--Decoder will trust its own decoding logic (ignore byte length). Decoder will start decoding the next tile right after the previous tile finished decoding (bitstream handling)
	17:16	**Reserved**
	15:8	**Reserved** Format: MBZ
	7	**Reserved**
	6	**Reserved**
	5	**Reserved**
	4	**Reserved**
	3:2	**Reserved**
	1	**Reserved**
	0	**VP9 Flush Handling** This modifies how flush is handled "0"--In VP9 mode, all unit done is accounted for before flush; "1"--In VP9 mode, only frame done is accounted for before flush; Value: 0 — Name: Flush Using Unit Done **[Default]** Value: 1 — Name: Flush Using Frame Done
4..5	31:0	**Reserved** Format: MBZ
5	31:0	**Reserved** Format: MBZ

HCP_QM_STATE

HCP_QM_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_QM_STATE command loads the custom HEVC quantization tables into local RAM and may be issued up to 20 times: 3x Colour Component plus 2x intra/inter plus 4x SizeID minus 4 for the 32x32 chroma components.
When the scaling_list_enable_flag is set to disable, the scaling matrix is still sent to the decoder, and with all entries programmed to the same value = 16.
This is a picture level state command and is issued in both encoding and decoding processes.

Dwords 2-17 form a table for the DCT coefficients, 4 8-bit coefficients/DWord.

- Size 4x4 for SizeID0, DWords 2-5.
- Size 8x8 for SizeID1/2/3, DWords 2-17.

SizeID 0 (Table 4-10)

4x4	[31:24]	[23:16]	[15:8]	[7:0]
DWord 2	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 3	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 4	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 5	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)

SizeID 1, 2, 3 (Table 4-11)

8x8	[31:24]	[23:16]	[15:8]	[7:0]	[31:24]	[23:16]	[15:8]	[7:0]
DWord 3,2	AC(0,7)	AC(0,6)	AC(0,5)	AC(0,4)	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 5,4	AC(1,7)	AC(1,6)	AC(1,5)	AC(1,4)	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 7,6	AC(2,7)	AC(2,6)	AC(2,5)	AC(2,4)	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 9,8	AC(3,7)	AC(3,6)	AC(3,5)	AC(3,4)	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)
DWord 11,10	AC(4,7)	AC(4,6)	AC(4,5)	AC(4,4)	AC(4,3)	AC(4,2)	AC(4,1)	AC(4,0)
DWord 13,12	AC(5,7)	AC(5,6)	AC(5,5)	AC(5,4)	AC(5,3)	AC(5,2)	AC(5,1)	AC(5,0)
DWord 15,14	AC(6,7)	AC(6,6)	AC(6,5)	AC(6,4)	AC(6,3)	AC(6,2)	AC(6,1)	AC(6,0)
DWord 17,16	AC(7,7)	AC(7,6)	AC(7,5)	AC(7,4)	AC(7,3)	AC(7,2)	AC(7,1)	AC(7,0)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

HCP_QM_STATE

	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 4h HCP_QM_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 10h; Name:
1	30:13	**Reserved** Format: MBZ
	12:5	**DC Coefficient** Format: U8 Specifies the 8-bit DC coefficient for SizeID 2 and 3. **Programming Notes** The DC Coefficient must be set to zero for SizeID 0 and 1. The DC Coefficient must be set to scaling_list_dc_coef_minus8 + 8 for SizeID 2 and 3.
	4:3	**Color Component** Format: U2 Encoder: When RDOQ is enabled, scaling list for all 3 color components must be same. So this field is set to always 0.

Value	Name
0	Luma
1	Chroma Cb
2	Chroma Cr
3	Reserved

HCP_QM_STATE

	2:1	**SizeID** Format: U2 **Value / Name / Description** 0 / 4x4 / 1 / 8x8 / 2 / 16x16 / 3 / 32x32 / (Illegal Value for Colour Component Chroma Cr and Cb.)
	0	**Prediction Type** Format: U1 **Value / Name** 0 / Intra 1 / Inter
2..17	511:0	**See Tables 4-10 and 4-11**

HCP_REF_IDX_STATE

HCP_REF_IDX_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a slice level command used in both encoding and decoding processes. For decoder, it is issued with the HCP_BSD_OBJECT command.

Unlike AVC, HEVC allows 16 reference idx entries in each of the L0 and L1 list for a progressive picture. Hence, a max total 32 reference idx in both lists together. The same when the picture is a field picture. Regardless the number of reference idx entries, there are only max 8 reference pictures exist at any one time. Multiple reference idx can point to the same reference picture and can optionally pic a top or bottom field, or frame.

For P-Slice, this command is issued only once, representing L0 list. For B-Slice, this command can be issued up to two times, one for L0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 12h HCP_REF_IDX_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 10h; Name:

HCP_REF_IDX_STATE

1	31:5	**Reserved** Format: MBZ
	4:1	**num_ref_idx_l[RefPicListNum]_active_minus1** Format: U4 num_ref_idx_l[*RefPicListNum*]_active_minus1 Value: [0-14]; Name:
	0	**RefPicListNum** Format: U1 Value 0: Reference Picture List 0 Value 1: Reference Picture List 1
2..17	31:16	**Reserved** Format: MBZ
	15	**list_entry_lX[i]: bottom_field_flag** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Value 0: Bottom Field Value 1: Top Field **Programming Notes** Not supported in encoder mode.
	14	**list_entry_lX[i]: field_pic_flag** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Value 0: Video Frame Value 1: Video Field **Programming Notes** Not supported in encoder mode.

HCP_REF_IDX_STATE

	13	**list_entry_lX[i]: LongTermReference** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. **Value** / **Name** 0 – Short term reference 1 – Long term reference
	12	**luma_weight_lX_flag[i]** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. **Value** / **Name** 0 – Default weighted prediction for luma 1 – Explicit weighted prediction for Luma
	11	**chroma_weight_lX_flag[i]** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. **Value** / **Name** 0 – Default weighted prediction for Chroma 1 – Explicit weighted prediction for Chroma
	10:8	**list_entry_lX[i]: Reference Picture Frame ID (RefAddr[0-7])** Format: U3 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. The reference picture frame ID identifies the reference picture associated with the base address defined in **Reference Picture Address (RefAddr[0-7])** in the HCP_PIPE_BUF_ADDR_STATE command.
	7:0	**list_entry_lX[i]: Reference Picture tb Value** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. clip(-128,127, CurrentPOC - RefPOC), where RefPOC is the POC value of the reference picture. 8-bit signed. See the "Derivation process for temporal luma motion vector prediction" in the HEVC standard.

HCP_SLICE_STATE

HCP_SLICE_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a slice level command used in both encoding and decoding processes. For decoder, it is issued with the HCP_BSD_OBJECT command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 14h HCP_SLICE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value / Name: 9h; 7h
1	31:25	**Reserved** Format: MBZ
	24:16	**SliceStartCtbY or (slice_start_lcu_y encoder)** Format: U9 Specifies the starting row address of the first coding tree block in the current slice.

HCP_SLICE_STATE

	15:9	**Reserved** Format: MBZ
	8:0	**SliceStartCtbX or (slice_start_lcu_x encoder)** Format: U9 Specifies the starting column address of the first coding tree block in the current slice.
2	31:25	**Reserved** Format: MBZ
	24:16	**NextSliceStartCtbY or (next_slice_start_lcu_y encoder)** Format: U9 Specifies the starting row address of the first coding tree block in the next slice. Must be set to zero when the current slice is the last slice of a picture. For the single slice per frame case, the only slice is also the last slice, so this parameter should be set to a number larger than the frame height (at least +1).
	15	**Reserved**
	14:9	**Reserved** Format: MBZ
	8:0	**NextSliceStartCtbX or (next_slice_start_lcu_x encoder)** Format: U9 Specifies the starting column address of the first coding tree block in the next slice. Must be set to zero when the current slice is the last slice of a picture. For the single slice per frame case, the only slice is also the last slice, so this parameter should be set to a number larger than the frame width (at least +1).
3	31:26	**Reserved** Format: MBZ
	25	**Reserved** Format: MBZ
	24	**Reserved** Format: MBZ
	23	**Reserved** Format: MBZ
	22	**Reserved**
	21:17	**slice_cr_qp_offset** Format: S4 For deblocking purpose, the pic and slice level cr qp offset must be provided separately.

HCP_SLICE_STATE

PAK needs to perform final_chroma_cr_qp_offset = pic_cr_qp_offset + slice_cr_qp_offset.

Value	Name
14h	-12
15h	-11
16h	-10
17h	-9
18h	-8
19h	-7
1Ah	-6
1Bh	-5
1Ch	-4
1Dh	-3
1Eh	-2
1Fh	-1
0h	0
1h	1
2h	2
3h	3
4h	4
5h	5
6h	6
7h	7
8h	8
9h	9
0Ah	10
0Bh	11
0Ch	12

Programming Notes
The valid value is from -12 to 12 (or 14h to 0Ch).

16:12 **slice_cb_qp_offset**

Format:	S4

For deblocking purpose, the pic and slice level cb qp offset must be provided separately.

PAK needs to perform final_chroma_cb_qp_offset = pic_cb_qp_offset + slice_cb_qp_offset.

HCP_SLICE_STATE

Value	Name
14h	-12
15h	-11
16h	-10
17h	-9
18h	-8
19h	-7
1Ah	-6
1Bh	-5
1Ch	-4
1Dh	-3
1Eh	-2
1Fh	-1
0h	0
1h	1
2h	2
3h	3
4h	4
5h	5
6h	6
7h	7
8h	8
9h	9
0Ah	10
0Bh	11
0Ch	12

Programming Notes
The valid value is from -12 to 12 (or 14h to 0Ch).

11:6 **SliceQp**

Format:	U6

Specifies the initial absolute value of QPy quantization parameter for the slice as defined in the Slice Header Semantics section of the HEVC standard.
This signifies only the magnitude of SliceQp. In 8 bit, SliceQp only goes from 0 to 51. But in 10 bit, it needs to go from -12 to 51. There is a sign bit specifies at bit [3] below.

HCP_SLICE_STATE

	5	**slice_temporal_mvp_enable_flag** Format: U1 **Programming Notes** Must be same for all the slices within a frame in encoder mode (follow spec)
	4	**dependent_slice_flag** Format: U1 Decoder only.
	3	**SliceQp Sign Flag** Format: U1 This specifes the sign bit of SliceQp. This is added for HEVC 10 bit. For 8 bit, SliceQp goes from 0 to 51 so this bit should be zero. In 10 bit, SliceQp goes from -12 to 51 and this bit can be set for negative value.
	2	**LastSliceofPic** Format: U1 This indicates the current slice is the very last slice of the current picture Value: 0 — Not the last slice of the picture Value: 1 — Last slice of the picture
	1:0	**slice_type** Format: U2 Value: 0 — B-slice Value: 1 — P-slice Value: 2 — I-slice Value: 3 — Illegal/Reserved
4	31:29	**Reserved** Format: MBZ
	28:26	**CollocatedRefIDX** Format: U3 Collocated Motion Vector Temporal Buffer Index.
	25:23	**MaxMergeIDX** Format: U3 MaxNumMergeCand = 5 - five_minus_max_num_merge_cand -1.

HCP_SLICE_STATE

Value	Name
0	0
1	1
2	2
3	3
4	4

Programming Notes
The valid value is from 0 to 4 (MaxNumMergeCand = 5 - five_minus_max_num_merge_cand -1)

22	**cabac_init_flag** Format: U1
21:19	**luma_log2_weight_denom** Format: U3
18:16	**ChromaLog2WeightDenom** Format: U3
15	**collocated_from_l0_flag** Format: U1
14	**isLowDelay** Format: U1 If the POCs of all pictures in both lists are less than the current POC, then set to one, else set to zero.
13	**mvd_l1_zero_flag** Format: U1 Decoder only.
12	**slice_sao_luma_flag** Format: U1
11	**slice_sao_chroma_flag** Format: U1
10	**slice_loop_filter_across_slices_enabled_flag** Format: U1
9	**Reserved** Format: MBZ
8:5	**slice_beta_offset_div2 or (final Beta_Offset_div2 Encoder)** Format: S3 Deblocking filter beta offset. Specified in 2's comp.

HCP_SLICE_STATE

		Value: [1101b,0011b] — Name: [-3,3]
	4:1	**slice_tc_offset_div2 or (final tc_offset_div2 Encoder)** Format: S3 Deblocking filter tc offset. Specified in 2's comp. Value: [1101b,0011b] — Name: [-3,3]
	0	**slice_header_disable_deblocking_filter_flag** Format: U1
5	31:16	**Reserved** Format: MBZ
	15:0	**SliceHeaderLength** Format: U16 Decoder only. Specifies the length in bytes of the slice header including the start code. The starting byte of the slice header in the bit stream buffer is indicated by the Indirect Data Start Address in the HCP_BSD_OBJECT command. The ending byte of the slice header in the same bit stream buffer is indicated by the last byte prior to the slice data (CABAC).
6	31:30	**Reserved** Format: MBZ
	29:26	**RoundInter** Format: U4

Value	Name
0h	+1/32
1h	+2/32
2h	+3/32
3h	+4/32
4h	+5/32 **[Default]**
5h	+6/32
6h	+7/32
7h	+8/32
8h	+9/32
9h	+10/32
Ah	+11/32

HCP_SLICE_STATE

		Bh: +12/32 Ch: +13/32 Dh: +14/32 Eh: +15/32 Fh: +16/32 **Programming Notes**: Encoder only feature
	25:24	**Reserved** Format: MBZ
	23:20	**RoundIntra** Format: U4 0h: +1/32 1h: +2/32 2h: +3/32 3h: +4/32 4h: +5/32 **[Default]** 5h: +6/32 6h: +7/32 7h: +8/32 8h: +9/32 9h: +10/32 Ah: +11/32 Bh: +12/32 Ch: +13/32 Dh: +14/32 Eh: +15/32 Fh: +16/32 **Programming Notes**: Encoder only feature
	19:0	**Reserved** Format: MBZ

HCP_SLICE_STATE

7 — 31:11 Reserved

Format:	MBZ

10 Header Insertion Enable

Format:	U1

Must be followed by the PAK Insertion Object Command to perform the actual insertion.

Value	Name	Description
0		No header insertion into the output bitstream buffer, before the current slice encoded bits.
1		Header insertion into the output bitstream buffer is present, and is before the current slice encoded bits.

Programming Notes
Must be always enabled. Encoder Only feature

9 SliceData Enable

Format:	U1

Must always be enabled.
Encoder only feature.

Value	Name	Description
0		No operation; no insertion.
1		Slice Data insertion by PAK Object Commands into the output bitstream buffer.

8 Tail Insertion Enable

Format:	U1

Must be followed by the PAK Insertion Object Command to perform the actual insertion.

Value	Name	Description
0		No tail insertion into the output bitstream buffer, after the current slice encoded bits.
1		Tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits. Restriction: Tail insertion is only possible at the end of frame but not in the middle (say slice end)

Programming Notes
Tail Insertion is allowed only at the end of last slice or last tile of a frame but not in the middle of frame. Also, no multiple tail insertions are allowed. Encoder only feature

7:3 Reserved

Format:	MBZ

HCP_SLICE_STATE

2 — EmulationByteSliceInsertEnable

Format:	U1

To have PAK outputting SODB or EBSP to the output bitstream buffer.

Value	Name
0	outputting RBSP
1	outputting EBSP

Programming Notes
Encoder Only feature

1 — CabacZeroWordInsertionEnable

Format:	U1

To pad the end of a SliceLayer RBSP to meet the encoded size requirement.

Value	Name	Description
0		No Cabac_Zero_Word Insertion.
1		Allow internal Cabac_Zero_Word generation and append to the end of RBSP (effectively can be used as an indicator for last slice of a picture, if the assumption is only the last slice of a picture needs to insert CABAC_ZERO_WORDs).

Programming Notes
Encoder Only feature

0 — Reserved

Format:	MBZ

8 — 31:29 — Reserved

Format:	MBZ

28:6 — Indirect PAK-BSE Data Start Offset (Write)

Format:	U23

This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the HCP PAK-BSE Object Base Address.

It is a cacheline-aligned address for the HEVC bitstream data.

For Write, there is no need to have a data length field. It is assumed the global memory upper bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access.

Value	Name
0-512MB	

HCP_SLICE_STATE

		Programming Notes Must be zero. Encoder Only feature
	5:0	**Reserved** Format: MBZ
9	31:16	**Reserved** Format: MBZ
	15:0	**Transformskip_lambda** Lambda value used in transform skip calculation **Programming Notes** Encoder Only feature
10	31:24	**Transformskip_numnonzerocoeffs_factor1** Multiplying factor with number of non-zero coefficients for non-Skip pipe calculation **Programming Notes** Encoder Only feature
	23:16	**Transformskip_numzerocoeffs_factor1** Multiplying factor with number of zero coefficients for non-Skip pipe calculation **Programming Notes** Encoder Only feature
	15:8	**Transformskip_numnonzerocoeffs_factor0** Multiplying factor with number of non-zero coefficients for skip pipe calculation **Programming Notes** Encoder Only
	7:0	**Transformskip_numzerocoeffs_factor0** Multiplying factor with number of zero coefficients for skip pipe calculation **Programming Notes** Encoder Only feature

HCP_SURFACE_STATE

HCP_SURFACE_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_SURFACE_STATE command is responsible for defining the frame buffer pitch and the offset of the chroma component.
This is a picture level state command and is shared by both encoding and decoding processes.
Note : Only NV12 and Tile Y are being supported for HEVC. Hence full pitch and interleaved UV is always in use. U and V Xoffset must be set to 0; U and V Yoffset must be 16-pixel aligned. This Surface State is not the same as that of the 3D engine and of the MFX pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 1h HCP_SURFACE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 1h — Name:
1	31:28	**Surface Id** Format: U4

HCP_SURFACE_STATE

Value	Name	Description
0h	HEVC: For current decoded Picture	8-bit uncompressed data
1h	Source Input Picture (encoder)	8-bit uncompressed data
2h	Prev Reference Picture	(VP9 only) Previous Reference
3h	Golden Reference Picture	(VP9 only) Golden Reference
4h	AltRef Reference Picture	(VP9 only) AltRef Reference

27:17 Reserved

Format:	MBZ

16:0 Surface Pitch Minus1

Format:	U17-1

This field specifies the surface pitch in (#Bytes - 1).

Programming Notes

- For tiled surfaces, the pitch must be a multiple of the tile width.
- If **Half Pitch for Chroma** is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.

For Y-tiled surfaces: Range = [127, 524287]->[128B,256KB] = [1 tile, 2048 tiles].

2 31:28 Surface Format

Format:	U4

Specifies the format of the surface.

Value	Name	Programming Notes
0h-2h	Reserved	
3h	P010Variant	P010Variant is a modified P010 format, >8 bit planar 420 with MSB together and LSB at an offset in x direction where the x-offset should be 32-bit aligned.
4h	PLANAR_420_8	
5h-Ch	Reserved	
Dh	P010	
Eh	P016	This is added for VP9 8./10/12 bit decode
Fh	Reserved	

27:15 Reserved

Format:	MBZ

14:0 Y Offset for U(Cb) in pixel

Format:	U15_Pixel_Row_Offset

This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled. This field is only used for PLANAR surface formats.

HCP_SURFACE_STATE		
		Programming Notes The following restrictions are applicable when Memory compression is enabled. • For PLANAR_420 surface formats, the alignment of this field follows the tile mode described in bits 14:13 of the **Memory Address Attributes** table. • TileY (legacy 4k) - 8 pixel aligned • TileYF (New 4k) - 64 pixel aligned • TileYS (64k) - 256 pixel aligned When Memory compression is not enabled, This field should be multiple of 8 pixels.

HCP_TILE_STATE

HCP_TILE_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 11h HCP_TILE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: Bh — Name:
1	31:10	**Reserved** Format: MBZ
	9:5	**NumTileColumnsMinus1** Format: U5 Specifies the number of tile columns in Ctbs per picture.
	4:0	**NumTileRowsMinus1** Format: U5 Specifies the number of tile rows in Ctbs per picture.

HCP_TILE_STATE		
2	31:24	**Ctb column position of tile column 3** Format: U8
	23:16	**Ctb column position of tile column 2** Format: U8
	15:8	**Ctb column position of tile column 1** Format: U8
	7:0	**Ctb column position of tile column 0** Format: U8
3	31:24	**Ctb column position of tile column 7** Format: U8
	23:16	**Ctb column position of tile column 6** Format: U8
	15:8	**Ctb column position of tile column 5** Format: U8
	7:0	**Ctb column position of tile column 4** Format: U8
4	31:24	**Ctb column position of tile column 11** Format: U8
	23:16	**Ctb column position of tile column 10** Format: U8
	15:8	**Ctb column position of tile column 9** Format: U8
	7:0	**Ctb column position of tile column 8** Format: U8
5	31:24	**Ctb column position of tile column 15** Format: U8
	23:16	**Ctb column position of tile column 14** Format: U8
	15:8	**Ctb column position of tile column 13** Format: U8
	7:0	**Ctb column position of tile column 12** Format: U8
6	31:24	**Ctb column position of tile column 19** Format: U8
	23:16	**Ctb column position of tile column 18**

HCP_TILE_STATE			
		Format:	U8
	15:8	**Ctb column position of tile column 17**	
		Format:	U8
	7:0	**Ctb column position of tile column 16**	
		Format:	U8
7	31:24	**Ctb row position of tile row 3**	
		Format:	U8
	23:16	**Ctb row position of tile row 2**	
		Format:	U8
	15:8	**Ctb row position of tile row 1**	
		Format:	U8
	7:0	**Ctb row position of tile row 0**	
		Format:	U8
8	31:24	**Ctb row position of tile row 7**	
		Format:	U8
	23:16	**Ctb row position of tile row 6**	
		Format:	U8
	15:8	**Ctb row position of tile row 5**	
		Format:	U8
	7:0	**Ctb row position of tile row 4**	
		Format:	U8
9	31:24	**Ctb row position of tile row 11**	
		Format:	U8
	23:16	**Ctb row position of tile row 10**	
		Format:	U8
	15:8	**Ctb row position of tile row 9**	
		Format:	U8
	7:0	**Ctb row position of tile row 8**	
		Format:	U8
10	31:24	**Ctb row position of tile row 15**	
		Format:	U8
	23:16	**Ctb row position of tile row 14**	
		Format:	U8
	15:8	**Ctb row position of tile row 13**	
		Format:	U8

HCP_TILE_STATE		
	7:0	**Ctb row position of tile row 12** Format: U8
11	31:24	**Ctb row position of tile row 19** Format: U8
	23:16	**Ctb row position of tile row 18** Format: U8
	15:8	**Ctb row position of tile row 17** Format: U8
	7:0	**Ctb row position of tile row 16** Format: U8
12	31:16	**Reserved** Format: MBZ
	15:8	**Ctb row position of tile row 21** Format: U8
	7:0	**Ctb row position of tile row 20** Format: U8

HCP_VP9_PAK_OBJECT

HCP_VP9_PAK_OBJECT

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Reserved**
	22:16	**Media Instruction Command** Default Value: 35h HCP_VP9_PAK_OBJECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 0h — Name:
1	31	**IsLastSB Flag (of Tile)** Format: Enable Value 0 — Name: False Value 1 — Name: True
	30	**Reserved** Format: MBZ
	29:24	**CU_count_minus1** Format: U6 Number of CUs in the current SB = CU_count_minus1 + 1. Minimum, there must be 1 CU in a SB.
	23:21	**Reserved** Format: MBZ

HCP_VP9_PAK_OBJECT

	20:0	**split_coding_unit_flag[x0][y0]** Format: U21 Bit 20: Split_flag_level0 Bit 19:16: Split_flag_level1 [19:16] is in raster order. Bit16 is for partition0 in raster order. Bit 15:12: Split_flag_level2_level1part3 Split flags for bit19 partition. [15:12] is in raster order. Bit12 is for partition0 in raster order. Bit 11:8: Split_flag_level2_level1part2 Split flags for bit18 partition. [11:8] is in raster order. Bit8 is for partition0 in raster order. Bit 7:4: Split_flag_level2_level1part1 Split flags for bit17 partition. [7:4] is in raster order. Bit4 is for partition0 in raster order. Bit 3:0: Split_flag_level2_level1part0 Split flags for bit16 partition. [3:0] is in raster order. Bit0 is for partition0 in raster order.
2	31:16	**Current SB Y Addr)** Format: U16
	15:0	**Current SB X Addr** Format: U16

HCP_VP9_PIC_STATE

HCP_VP9_PIC_STATE

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 30h HCP_VP9_PIC_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value / Name / Programming Notes: Bh – Decoder DW Length – Only Up to DW12 should be programmed for decoder 1Eh – Encoder DW Length – All DWs should be programmed for encoder
1	31:30	**Reserved** Format: MBZ
	29:16	**Frame Height In Pixels Minus 1** Format: U14 Specifies the height of the decoded picture in units of 8 pixels, which is the minimum coding block size. The decoded picture height in units of luma samples equals (FrameHeightInMinBlocksMinus1+ 1) *8 – 1 *For Encoder Partial SB:* *Kernel, on last SB (at picture edges), splits the SB into as small as possible CUs to have picture boundaries aligned to CU boundary.*

HCP_VP9_PIC_STATE

Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively.
Driver sets up a SB aligned (both in X/Y direction) surface.

Value	Name
[4096,16383]	4K_TO_16K

Programming Notes

Decoder supports 16K image and 8K video. Encoder only supports upto 8K.

Value	Description
0-6	Invalid (multiple of 8 pixels)
7-8191	8-8K Pixels (Decoder and Encoder)
8192-16383	8K-16K Pixels (Decoder only)

15:14 Reserved

Format:	MBZ

13:0 Frame Width In Pixels Minus 1

Format:	U14

Specifies the width of the decoded picture in units of minimum coding block size. The decoded picture width in units of luma samples equals (FrameWidthInMinBlocksMinus1+ 1) *8 – 1 This should be programmed to a multiple of 8 pixels minus 1.
For Encoder Partial SB:
Kernel, on last SB (at picture edges), splits the SB into as small as possible CUs to have picture boundaries aligned to CU boundary.
Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively.
Driver sets up a SB aligned (both in X/Y direction) surface.

Value	Name
[4096,16383]	4K_TO_16K

Programming Notes

Decoder supports 16K image and 8K video. Encoder only supports upto 8K.

Value	Description
0-6	Invalid (multiple of 8 pixels)
7-8191	8-8K Pixels (Decoder and Encoder)
8192-16383	8K-16K Pixels (Decoder only)

2 31 Segment ID StreamIn Enable

Format:	Enable

Indicates SegmentID from previous frame needs to be streamIn for Segment ID prediction

Value	Name
0	Disable
1	Enable

HCP_VP9_PIC_STATE

Programming Notes
Deocder Only: SegmentIDStreamInEnable = error_resilient_mode OR intra_only OR VP9_KEY_FRAME

30 **Segment ID StreamOut Enable**

Format:	Enable

Indicates SegmentID of current frame needs to be streamOut for next frame

Value	Name
0	Disable
1	Enable

Programming Notes
Deocder Only: SegmentIDStreamOutEnable = segementation_enabled AND segmentation_update_map

29 **Lossless Mode**

Format:	Enable

This bitSet to indicate lossless coding mode.
In encoder mode, software has to set tx_mode to 4x4only and all tu_size's in CU record as 4x4 for entire frame. Software also has to program such that final_qindex=0 and final_filter_level=0 following the Quant Scale and Filter Level Table in Segmentation State section. Hardware forces Hadamard Tx when this bit is set. When Lossless Mode is on, BRC has to be off.

Value	Name
0	Normal Mode
1	Loless Mode

28 **Segmentation Temporal Update**

Format:	Enable

Indicates whether segID is decoding from bitstream or predicted from previous frame.
In encoder Mode it should use either from previous frame or streamIn

Value	Name
0h	Decode segID from bitstream
1h	Get segID either from bitstream or from previous frame

Programming Notes
Decoder Only: For KEY_FRAME or INTRA_ONLY frame, this bit should be set to "0". Note: Driver should override this flag to "0" in KEY_FRAME or INTRA_ONLY frame even if this

HCP_VP9_PIC_STATE

		bit decoded from bitstream is different. This is for hardware optimization. This override does not affect bitstream decoding other than uncompressed header.
	27	**Segmentation Update Map** Format: Enable Indicates how hardware determines segmentation ID 0h: Intra block: segment ID is zero Inter block: get segment ID from previous frame (streamIN) 1h: Intra block: decode segment ID from bitstream. Inter block: determins from segmentation_temporal_update setting
	26	**Segmentation Enabled** Format: Enable Indicate if segementation is enabled or not 0h: All blocks are implied to belong to segment 0 1h: SegID determination depends on segmentation_update_map setting
	25:23	**Sharpness Level** Format: U3 Specify the sharpness level, as one of regular deblocking strength control. Programming Notes: Set to 0 to disable the use of sharpness control
	22:17	**Filter Level** Format: U6 Specify the Filter level, as one of deblocking strength control Programming Notes: Set to 0 to disable the use of level control
	16	**Frame Parallel Decoding Mode** Indicates if parallel decoding mode is enabled. This bit should come from Uncompressed header. Together with Error Resilient mode, they decide the value of AdaptProbabilityFlag. 0: Disable 1: Enable
	15	**Error Resilient Mode** Indicates if error resilient mode is enabled. This bit should come from Uncompressed header.When error resilient is 1, Frame Parallel Decoding Mode will be 1, and Refresh Frame Context will be 0.When error resilient is 0, Frame Parallel Decoding Mode and Refresh Frame Context read from bit stream.Together with Frame Parallel Decoding mode, they decide the value of AdaptProbabilityFlag.

HCP_VP9_PIC_STATE

Value	Name
0	Disable
1	Enable

Programming Notes

14 **Refresh Frame Context**

Indicates if Frame Context should be refresh. This bit should come from Uncompressed header

Value	Name
0	Disable
1	Enable

Programming Notes
Decoder Only

13 **Last Frame Type**

It indicates the frame type of previous frame (Key or Non-Key Frame)

Value	Name
0	Key Frame
1	Non Key Frame

Programming Notes
Not used in Encoder Mode

12 **Selectable TX Mode**

Format:	U1

Indicates if tx_mode is selectable

Value	Name
0	Encoder does not pack tu_size into bitstream. This helps reduce bitstream size further.
1	Encoder packs tu_size into bitstream.

Programming Notes
HW always picks tu_size from CU record of pak_obj. SW responsibility to set tu_size correct.

11 **Hybrid Prediction Mode**

Format:	U1

Indicates if comp_pred_mode is hybrid

Value	Name
0	comp_prediction_mode!=HYBRID, Encoder does not pack comp_pred_mode [interpred_comp in pak_obj] into bitstream.

HCP_VP9_PIC_STATE

	1	comp_prediction_mode==HYBRID, Encoder packs comp_pred_mode into bitstream. This helps reduce bitstream size further.

10 Use Prev in Find MV References

Format:	Enable

0: Temporal MV buffer is not available for MV prediction
1: Temporal MV buffer is available for MV prediction
This is set to 0 when:
• The last picture has a different size
• Current picture is error-resilient mode
• Current picture is intra_only, or keyframe
• Last picture was intra_only or keyframe
• Last picture was not a displayed picture.

9:7 Ref Frame Sign Bias[0..2]

Format:	U3

Reference Frame sign bias (not including intra reference)
Bit[7] – Sign Bias of Last Frame
Bit[8] – Sign Bias of Golden Frame
Bit[9] – Sign Bias of AltRef Frame

6:4 Mcomp Filter Type

Format:	U3

Indicate Motion Compensation Filter type.
If set to 4, encoder uses modes in pak_obj command.

Value	Name
0h	Eight-tap
1h	Eight-tap-Smooth
2h	Eight-tap-Sharp
3h	Bilinear
4h	Switchable

3 Allow Hi Precision MV

Format:	Enable

Indicate high precision mode for Motion Vector prediction

Value	Name
0h	Normal mode
1h	High Precision mode

HCP_VP9_PIC_STATE

2 — IntraOnly Flag

Format:	Enable

Indicates intra-only for inter pics. MBZ for keyframes.

Programming Notes

Used for Non-displayable picture; SW responsibility to make sure no Inter block in pak_obj of this frame

Value	Description
0	Inter Frame use both intra/inter-blocks
1	Inter frame use only inta-blocks

1 — Adapt Probabilities Flag

Format:	Enable

Indicates that the probabilities used to decode this frame should be adapted

Value	Name
0h	0: Do not adapt (error resilient or frame_parallel_mode are set)
1h	1: Adapt (not error resilient and not frame_ parallel_mode)

Programming Notes

Only forward adaptation is supported in encoder mode.

0 — Frame Type

Format:	U1

Specifies the VP9 frame type

Value	Name
0h	Key Frame
1h	Inter Frame

3 — 31:28 — Profile Level

Format:	U4

This indicates VP9 Profile level from bitstream

Value	Name	Description
0	Profile_0	Profile 0 only supports 8 bit 420 only
2	Profile_2	Profile 2 only supports 10 bits 420 only

27:24 — BitDepthMinus8

Format:	U4

This indicates the bitdepth (minus 8) of the pixels

Value	Name	Programming Notes
0	Bitdepth_8	It indicates pixel bitdepth is 8. Only profile 0 is allowed in this mode.
2	Bitdepth_10	It indicates pixel bitdepth is 10. Only profile 2 is allowed in this mode.
4	Bitdepth_12	It indicates pixel bitdepth is 12. Only profile 2 is allowed in this mode.

HCP_VP9_PIC_STATE

23:22 Chroma Sampling Format

Format:	U2

This indicates the chroma sampling format of the bitstream

Value	Name	Programming Notes
0	Format_420	Chroma Format 420, supported by profile 0 and 2

Programming Notes
Currenly only 420 is supported (in profile 0 and 2). All other modes are not valid. Value 0: Format 420: Profile 0 and 2 only: Supported Value 1: Format 422: Profile 1 and 3 only: Currently NOT supported Value 2: Format 444: Profile 1 and 3 only: Currently NOT supported

21 Reserved

Format:	MBZ

20:10 Reserved

Format:	MBZ

9:8 Log2 Tile Row

Format:	U2

This indicates the number of tile rows (log2).

Value	Name
0h	1 Tile Row
1h	2 Tile Row
2h	4 Tile Row

Programming Notes
In encoder mode, create tiles either in horizontal or vertical direction but not in both directions

7:4 Reserved

Format:	MBZ

3:0 Log2 Tile Column

Format:	U4

This indicates the number of tile rows (log2).

Value	Name	Programming Notes
0h	1 Tile Column	
1h	2 Tile Column	
2h	4 Tile Column	In encoder mode, create tiles either in horizontal or vertical direction but not in both directions
3h	8 Tile Column	

HCP_VP9_PIC_STATE

		4h: 16 Tile Column 5h: 32 Tile Column 6h: 64 Tile Column **Programming Notes** In encoder mode, create tiles either in horizontal or vertical direction but not in both directions
4	31:16	**Horizontal Scale Factor for LAST** Format: U2.14 This indicates the scaling factor between current frame and the last reference frame Set to (LastWidth * 2^14)/CurrentWidth
	15:0	**Vertical Scale Factor for LAST** Format: U2.14 This indicates the scaling factor between current frame and the last reference frame Set to (LastHeight * 2^14)/CurrentHeight
5	31:16	**Horizontal Scale Factor for GOLDEN** Format: U2.14 This indicates the scaling factor between current frame and the golden reference frame Set to (LastWidth * 2^14)/CurrentWidth
	15:0	**Vertical Scale Factor for GOLDEN** Format: U2.14 This indicates the scaling factor between current frame and the golden reference frame Set to (LastHeight * 2^14)/CurrentHeight"
6	31:16	**Horizontal Scale Factor for ALTREF** Format: U2.14 This indicates the scaling factor between current frame and the altref reference frame Set to (LastWidth * 2^14)/CurrentWidth
	15:0	**Vertical Scale Factor for ALTREF** Format: U2.14 This indicates the scaling factor between current frame and the altref reference frame Set to (LastHeight * 2^14)/CurrentHeight

HCP_VP9_PIC_STATE

7	31:30	**Reserved** Format: MBZ
	29:16	**Last Frame Hieght In Pixels Minus 1** Format: U14 Specifies the height of the Last picture in units of pixels. The Last picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
	15:14	**Reserved** Format: MBZ
	13:0	**Last Frame Width In Pixels Minus 1** Format: U14 Specifies the width of the Last picture in units of pixels. The Last picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
8	31:30	**Reserved** Format: MBZ
	29:16	**Golden Frame Hieght In Pixels Minus 1** Format: U14 Specifies the height of the Last picture in units of pixels. The Golden picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
	15:14	**Reserved** Format: MBZ
	13:0	**Golden Frame Width In Pixels Minus 1** Format: U14 Specifies the width of the Last picture in units of pixels. The Golden picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
9	31:30	**Reserved** Format: MBZ

HCP_VP9_PIC_STATE

	29:16	**Altref Frame Hieght In Pixels Minus 1** Format: U14 Specifies the height of the Last picture in units of pixels. The Altref picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
	15:14	**Reserved** Format: MBZ
	13:0	**Altref Frame Width In Pixels Minus 1** Format: U14 Specifies the width of the Last picture in units of pixels. The Altref picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1) Value: 0-16383 — Description: 1-16K Pixels (encoder only goes to 8K)
10	31:16	**First Partition Size in Bytes [15:0]** Format: U16 Specifies the number of bytes taken up by the first partition size which handle the probability updates **Programming Notes** Only used by Decoder
	15:8	**Reserved** Format: MBZ
	7:0	**Uncompressed Header Length in Bytes [7:0]** Format: U8 Specifies the number of bytes taken up by the uncompressed frame header. **Programming Notes** this field is used by decoder only
11	31:4	**ChickenBits [31:4]** Format: MBZ This field is reserved to be used for chicken bits
	3	**Chroma Fractional Calculation Modified** This changes how Chroma Fractional Calculation is handled in MC "0"--Default version (current); "1"--Matched an earlier version of ref decoder **Value** — **Name**

HCP_VP9_PIC_STATE

		0 Current **[Default]** 1 Outdated
	2	**MV Clamp Disable** This modifies how hardware handles MV clamping logic "0"--MV clamping logic is enabled; "1"--MV clamping logic is disabled Not used in Encoder Mode **Value** / **Name** 0 Enable **[Default]** 1 Disable
	1	**Motion Comp Scaling Chicken Bit** This modifies how VP9 Motion Comp to handle scaling x/y calculation. 0 -- Pre-Dec13 version; 1--Post-Dec13 version **Value** / **Name** 0 Pre-Dec13 Version 1 Post-Dec13 version **[Default]**
	0	**ChickenBits[0]** Format: MBZ This field is reserved to be used for chicken bits
12	31:0	**ChickenBits [63:32]** Format: MBZ Value / Description 0 Only valid value 0x1-0xFFFFFFFF Invalid
13	31:26	**Reserved** Format: MBZ
	23:16	**Base Q Index (Same as Luma AC)** Format: U8 Added to Delta Q index of Segment Valid Values : 0..255 **Programming Notes** Encoder Only
	15:0	**Compressed header BIN count** Format: U16 Number of bins compressed header This field is fixed insideHW **Programming Notes**

HCP_VP9_PIC_STATE

		Encoder Only
14	31:21	**Reserved** Format: MBZ
	20:16	**Luma DC Q Index Delta** Format: S4 QP delta value for Luma DC Valid Values : -15..15 **Programming Notes** Encoder Only
	15:13	**Reserved** Format: MBZ
	12:8	**ChromaDC_QindexDelta** Format: S4 QP Delta Value For Chroma DC Valid Values : -15..15 **Programming Notes** Encoder Only
	7:5	**Reserved** Format: MBZ
	4:0	**ChromaAC_QindexDelta** Format: S4 QP Delta Value For Chroma AC Valid Values : -15..15 **Programming Notes** Encoder Only
15	31	**Reserved** Format: MBZ
	30:24	**LF_ref_delta3** Format: S6 Loop filter level delta3 value; valid range -63...63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	23	**Reserved** Format: MBZ

HCP_VP9_PIC_STATE

	22:16	**LF_ref_delta2** Format: S6 Loop filter level delta2 value; valid range -63...63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	15	**Reserved** Format: MBZ
	14:8	**LF_ref_delta1** Format: S6 Loop filter level delta1 value; valid range -63...63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	7	**Reserved** Format: MBZ
	6:0	**LF_ref_delta0** Format: S6 Loop filter level delta0 value; valid range -63...63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
16	31:15	**Reserved** Format: MBZ
	14:8	**LF Mode Delta 1** Format: S6 Loop filter level mode1 value; valid range -63...63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only

HCP_VP9_PIC_STATE

	7	**Reserved** Format: MBZ
	6:0	**LF Mode Delta 0** Format: S6 Loop filter level mode1 value; valid range -63…63 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
17	31:16	**BitOffsetForLFModeDelta** Format: U16 Offset from starting position of output bitstream in bits where LFModeDelta should be inserted. In BRC mode, always insert LFModeDelta. {This implies uncompressed header should have: mode_ref_delta_enabled=1 and mode_ref_delta_update=1} and The uncompressed header starting from this offset **BitOffsetForLFModeDelta** has to have the following 16 bits format: {Start here: 1'b1, mode_delta_0[6:0], 1'b1, mode_delta_1[6:0]} and **BitOffsetForLFModeDelta = BitOffsetForLFRefDelta + 32.** **Encoder Only**
	15:0	**BitOffsetForLFRefDelta** Format: U16 Offset from starting position of output bitstream in bits where LFRefDelta should be inserted. In BRC mode, always insert LFRefDelta. {This implies uncompressed header should have: mode_ref_delta_enabled=1 and mode_ref_delta_update=1} and The uncompressed header starting from this offset **BitOffsetForLFRefDelta** has to have the following 32 bits format: {Start here: 1'b1, ref_delta_0[6:0], 1'b1, ref_delta_1[6:0], 1'b1, ref_delta_2[6:0], 1'b1, ref_delta_3[6:0]}. Encoder Only
18	31:16	**BitOffsetForLFLevel** Format: U16 Offset from starting position of output bitstream in bits where LFLevel should be inserted. Encoder Only
	15:0	**BitOffsetForQindex** Format: U16 Offset from starting position of output bitstream in bits where Qindex should be inserted. Encoder Only

HCP_VP9_SEGMENT_STATE

HCP_VP9_SEGMENT_STATE

Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 32h HCP_VP9_SEGMENT_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 6h — Name:
1	31:3	**Reserved** Format: MBZ
	2:0	**Segment ID** Format: U3 The segment ID of the DWORDS following this one
2	31:4	**Reserved** Format: MBZ
	3	**Segment Reference Enabled** Format: Enable For encoder: When "Segment Reference Enabled" is set to 1, Software (Kernel) needs to program all PUs of this segment accordingly. This means: CU record PU refframe0="Segment Reference" bit[2:1], refframe1=0, and interpred_comp=single.

HCP_VP9_SEGMENT_STATE

	2:1	**Segment Reference** Format: U2 Indicates reference frame for this segment. **Programming Notes** If the current frame is a KEY frame or INTRA_ONLY frame, this field should be set to INTRA for all segments.
	0	**Segment Skipped** Format: Enable Indicates skip mode for this segment. **Programming Notes** If set to 1, all delta coefficients and MVs within CU of this segement should be forced to zero in HW. No block less than 8x8 size allowed segmentSkipped If set to 0, skipcoeff_flag should be coded as normal
3	31:30	**Reserved** Format: MBZ
	29:24	**FilterLevelRef1Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref1 (Last Frame) and Mode 1.
	23:22	**Reserved** Format: MBZ
	21:16	**FilterLevelRef1Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref1 (Last Frame) and Mode 0.
	15:14	**Reserved** Format: MBZ
	13:8	**FilterLevelRef0Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref0 (Last Frame) and Mode 1.
	7:6	**Reserved** Format: MBZ
	5:0	**FilterLevelRef0Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref0 (Last Frame) and Mode 0.

HCP_VP9_SEGMENT_STATE

4	31:30	**Reserved** Format: MBZ
	29:24	**FilterLevelRef3Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref3 (Last Frame) and Mode 1.
	23:22	**Reserved** Format: MBZ
	21:16	**FilterLevelRef3Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref3 (Last Frame) and Mode 0.
	15:14	**Reserved** Format: MBZ
	13:8	**FilterLevelRef2Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref2 (Last Frame) and Mode 1.
	7:6	**Reserved** Format: MBZ
	5:0	**FilterLevelRef2Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref2 (Last Frame) and Mode 0.
5	31:16	**Luma AC Quant Scale (Decode mode Only)** Format: S15 Indicates final value of Luma AC Quantized Scale value.
	15:0	**Luma DC Quant Scale (Decode mode Only)** Format: S15 Indicates final value of Luma DC Quantized Scale value.

HCP_VP9_SEGMENT_STATE		
6	31:16	**Chroma AC Quant Scale (Decode mode Only)** Format: S15 Indicates final value of Chroma AC Quantized Scale value.
	15:0	**Chroma DC Quant Scale (Decode mode Only)** Format: S15 Indicates final value of Chroma DC Quantized Scale value.

HCP_WEIGHTOFFSET_STATE

HCP_WEIGHTOFFSET_STATE

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This slice level command is issued in both the encoding and decoding processes, if the weighted_pred_flag or weighted_bipred_flag equals one. If zero, then this command is not issued.
Weight Prediction Values are provided in this command. Only Explicit Weight Prediction is supported in encoder.
For P-Slice, this command is issued only once together with HCP_REF_IDX_STATE Command for L0 list. For B-Slice, this command can be issued up to two times together with HCP_REF_IDX_STATE Command, one for L0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 13h HCP_WEIGHTOFFSET_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 20h — Name: 32
1	31:1	**Reserved** Format: MBZ

HCP_WEIGHTOFFSET_STATE

	0	**RefPicListNum** Format: U1 Value / Name 0 – Reference Picture List 0 1 – Reference Picture List 1
2..17	31:16	**Reserved** Format: MBZ
	15:8	**luma_offset_lX[i]** Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for luma is enabled, otherwise must be zero. **Programming Notes** This (combined with it MSbyte above) shall be in the range of −WpOffsetHalfRange$_Y$ to WpOffsetHalfRange$_Y$ − 1, where WpOffsetHalfRange$_Y$ = 1 « (high_precision_offsets_enabled_flag ? (BitDepth$_Y$ − 1) : 7)
	7:0	**delta_luma_weight_lX[i]** Format: S7 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for luma is enabled, otherwise must be zero. **Programming Notes** When luma_weight_l0_flag[i] is equal to 1, the value of delta_luma_weight_l0[i] shall be in the range of −128 to 127, inclusive.
18..33	31:24	**ChromaOffsetLX [i][1]** Where X is the RefPicListNum and i is the list entry number 0 through 15. DW 18corresponds to i=0, DW 33 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero. **Programming Notes** This (combined with its MSbyte below) shall be in the range of -WpOffsetHalfRangeC to (WpOffsetHalfRangeC - 1), inclusive WpOffsetHalfRangeC = 1 « (high_precision_offsets_enabled_flag ? (BitDepthC - 1) : 7)

HCP_WEIGHTOFFSET_STATE

	23:16	**delta_chroma_weight_lX[i][1]** Format: S7 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW 18 corresponds to i=0, DW 33 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero. **Programming Notes** This shall be in the range of −128 to 127, inclusive
	15:8	**ChromaOffsetLX[i][0]** Where X is the RefPicListNum and i is the list entry number 0 through 15. DW 18 corresponds to i=0, DW 33 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero. **Programming Notes** This (combined with its MSbyte below) shall be in the range of -WpOffsetHalfRangeC to (WpOffsetHalfRangeC - 1), inclusive WpOffsetHalfRangeC = 1 « (high_precision_offsets_enabled_flag ? (BitDepthC - 1) : 7)
	7:0	**delta_chroma_weight_lX[i][0]** Format: S7 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW 18 corresponds to i=0, DW 33 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero. **Programming Notes** This shall be in the range of −128 to 127, inclusive

HEVC_VP9_RDOQ_STATE

HEVC_VP9_RDOQ_STATE

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h Format: Opcode PARALLEL_VIDEO_PIPE
	28:27	**Pipeline** Default Value: 2h Format: Opcode MFX_COMMON
	26:23	**Opcode** Default Value: 7h Format: Opcode Codec/Engine Name = HCP = 7h
	22:21	**SubOpA** Default Value: 0h Format: Opcode
	20:16	**SubOpB** Default Value: 8h Format: Opcode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Total Length - 2 Value: 3Ch — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31	**Chicken bit to disable HTQ performance fix0** set to disable performance optimizations done by doubling LNZ and OSR storage Set to 1, to go back to single LNZ and OSR (no optimization) Set to 0, to use double buffer LNZ and OSR

HEVC_VP9_RDOQ_STATE		
	30	**Chicken bit to disable HTQ performance fix1** set to disableperformance optimization done to save 1clk while switching from EBB to GTRAM storage.This is critical for IntraLoop performance Set to 1, disable optimization Set to 0, enable optimization
	29:0	**Reserved** Format: MBZ
2	31:16	**Lambda_Intra_Luma1** Lambda value for Intra Luma component of QP=1 for HEVC Lambda value for Intra Luma in segment 1 for VP9
	15:0	**Lambda_Intra_Luma0** Lambda value for Intra Luma component of QP=0 for HEVC Lambda value for Intra Luma in segment 0 for VP9
3	31:16	**Lambda_Intra_Luma3** Lambda value for Intra Luma component of QP=3 for HEVC Lambda value for Intra Luma in segment 3 for VP9
	15:0	**Lambda_Intra_Luma2** Lambda value for Intra Luma component of QP=2 for HEVC Lambda value for Intra Luma in segment 2 for VP9
4	31:16	**Lambda_Intra_Luma5** Lambda value for Intra Luma component of QP=5 for HEVC Lambda value for Intra Luma in segment 5 for VP9
	15:0	**Lambda_Intra_Luma4** Lambda value for Intra Luma component of QP=4 for HEVC Lambda value for Intra Luma in segment 4 for VP9
5	31:16	**Lambda_Intra_Luma7** Lambda value for Intra Luma component of QP=7 for HEVC Lambda value for Intra Luma in segment 7 for VP9
	15:0	**Lambda_Intra_Luma6** Lambda value for Intra Luma component of QP=6 for HEVC Lambda value for Intra Luma in segment 6 for VP9
6..33	31:16	**Lambda_Intra_Luma9,11, ...,61, 63(odd number)** Lambda value for Intra Luma component of QP=9, 11, ..., 61, 63 (odd number) for HEVC
	15:0	**Lambda_Intra_Luma8, 10, ..., 48, 50 (even number)** Lambda value for Intra Luma component of QP=8, 10, ..., 60, 62 (even number) for HEVC
34	31:16	**Lambda_Intra_Chroma 1** Lambda value for Intra Chroma component of QP=1 for HEVC'Lambda value for Intra Chroma in segment 1 for VP9

HEVC_VP9_RDOQ_STATE		
	15:0	**Lambda_Intra_Chroma 0** Lambda value for Intra Chroma component of QP=0 for HEVC'Lambda value for Intra Chroma in segment 0 for VP9
35	31:16	**Lambda_Intra_Chroma 3** Lambda value for Intra Chroma component of QP=3 for HEVC'Lambda value for Intra Chroma in segment 3 for VP9
	15:0	**Lambda_Intra_Chroma 2** Lambda value for Intra Chroma component of QP=2 for HEVC'Lambda value for Intra Chroma in segment 2 for VP9
36	31:16	**Lambda_Intra_Chroma 5** Lambda value for Intra Chroma component of QP=5 for HEVC'Lambda value for Intra Chroma in segment 5 for VP9
	15:0	**Lambda_Intra_Chroma 4** Lambda value for Intra Chroma component of QP=4 for HEVC'Lambda value for Intra Chroma in segment 4 for VP9
37	31:16	**Lambda_Intra_Chroma 7** Lambda value for Intra Chroma component of QP=7 for HEVC'Lambda value for Intra Chroma in segment 7 for VP9
	15:0	**Lambda_Intra_Chroma 6** Lambda value for Intra Chroma component of QP=6 for HEVC'Lambda value for Intra Chroma in segment 6 for VP9
38..65	31:16	**Lambda_Intra_Chroma 9, 11, ..., 49, 51 (odd number)** Lambda value for Intra Chroma component of QP=9, 11, ..., 61, 63 (odd number) for HEVC
	15:0	**Lambda_Intra_Chroma 8, 10, ..., 48, 50 (even number)** Lambda value for Intra Chroma component of QP=8, 10, ..., 60, 62 (even number) for HEVC
66	31:16	**Lambda_Inter_Luma1** Lambda value for Inter Luma component of QP=1 for HEVC'Lambda value for Inter Luma in segment 1 for VP9
	15:0	**Lambda_Inter_Luma0** Lambda value for Inter Luma component of QP=0 for HEVC'Lambda value for Inter Luma in segment 0 for VP9
67	31:16	**Lambda_Inter_Luma3** Lambda value for Inter Luma component of QP=3 for HEVC'Lambda value for Inter Luma in segment 3 for VP9
	15:0	**Lamba_Inter_Luma2** Lambda value for Inter Luma component of QP=2 for HEVC'Lambda value for Inter Luma in segment 2 for VP9
68	31:16	**Lambda_Inter_Luma5** Lambda value for Inter Luma component of QP=5 for HEVC'Lambda value for Inter Luma in segment 5 for VP9

HEVC_VP9_RDOQ_STATE		
	15:0	**Lambda_Inter_Luma4** Lambda value for Inter Luma component of QP=4 for HEVC'Lambda value for Inter Luma in segment 4 for VP9
69	31:16	**Lambda_Inter_Luma7** Lambda value for Inter Luma component of QP=7 for HEVC'Lambda value for Inter Luma in segment 7 for VP9
	15:0	**Lambda_Inter_Luma6** Lambda value for Inter Luma component of QP=6 for HEVC'Lambda value for Inter Luma in segment 6 for VP9
70..97	31:16	**Lambda_ Inter _Luma9, 11, ..., 49, 51 (odd number)** Lambda value for Inter Luma component of QP= Luma9, 11, ..., 61, 63 (odd number) for HEVC
	15:0	**Lambda_Intra_ Inter 8, 10, ..., 48, 50 (even number)** Lambda value for Inter Luma component of QP=8, 10, ..., 60, 62 (even number) for HEVC
98	31:16	**Lambda_Inter_ Chroma 1** Lambda value for Inter Chroma component of QP=1 for HEVC'Lambda value for Inter Chroma in segment 1 for VP9
	15:0	**Lambda_ Inter _Chroma0** Lambda value for Inter Chroma component of QP=0 for HEVC'Lambda value for Inter Chroma in segment 0 for VP9
99	31:16	**Lambda_Inter _ Chroma 3** Lambda value for Inter Chroma component of QP=3 for HEVC'Lambda value for Inter Chroma in segment 3 for VP9
	15:0	**Lambda_ Inter _ Chroma 2** Lambda value for Inter Chroma component of QP=2 for HEVC'Lambda value for Inter Chroma in segment 2 for VP9
100	31:16	**Lambda_ Inter _ Chroma 5** Lambda value for Inter Chroma component of QP=5 for HEVC'Lambda value for Inter Chroma in segment 5 for VP9
	15:0	**Lambda_ Inter _ Chroma 4** Lambda value for Inter Chroma component of QP=4 for HEVC'Lambda value for Inter Chroma in segment 4 for VP9
101	31:16	**Lambda_ Inter _ Chroma 7** Lambda value for Inter Chroma component of QP=7 for HEVC'Lambda value for Inter Chroma in segment 7 for VP9
	15:0	**Lambda_ Inter _ Chroma 6** Lambda value for Inter Chroma component of QP=6 for HEVC'Lambda value for Inter Chroma in segment 6 for VP9
102..129	31:16	**Lambda_ Inter _ Chroma 9, 11, ..., 49, 51 (odd number)** Lambda value for Inter Chroma component of QP=9, 11, ..., 61, 63 (odd number) for HEVC
	15:0	**Lambda_Inter_ Chroma 8, 10, ..., 48, 50 (even number)** Lambda value for Inter Chroma component of QP=8, 10, ..., 60, 62 (even number) for HEVC

HI8DS Render Target Write MSD

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

17:14 Message Type

Default Value:	0Ch
Format:	Opcode

Render Target Write message

13 Per-Sample PS Enable

Format:	Enable

If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.

Programming Notes

This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.

When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.

When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).

When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.

12 Last Render Target Select

Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set.

In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 Slot Group Select

Format:	MDC_RT_SGS

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 3h Format: Opcode SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

HUC_CFG_STATE

HUC_CFG_STATE

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_CFG_STATE command is used to force the P24C (MinuteIA) into a reset condition as well as forcing it out of a reset condition. This command is not normally required since the hardware will handle placing the P24C into a reset condition and releasing it from reset, but there may be conditions that require a forced reset.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 3h HUC_CFG_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 0h — Name:
1	31:1	**Reserved** Format: MBZ

HUC_CFG_STATE

	0	**P24C (MinutelA)** Format: U1 **Value** / **Name** / **Description** 0 / Normal operation / No reset. 1 / Force reset /

HUC_DMEM_STATE

HUC_DMEM_STATE

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_DMEM_STATE command is used to fetch the HUC data from the graphics memory and load it into the HUC 96KB L2 storage RAM. The HUC_DMEM_STATE specifies the data source base address in graphics memory.
When the HUC_DMEM_STATE command is received, the data is loaded by the HUC DMA into the 96KB L2 storage RAM at the location provided in the HUC_DMEM_STATE command. This command also specifies the length of the data, which is specified in bytes but must be in increments of 64 byte cache lines.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 2h HUC_DMEM_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 4h; Name:
1..2	63:0	**HUC Data Source Base Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned HUC data source base address in graphics memory.

HUC_DMEM_STATE		
3	31:0	**HUC Data Source**
4	31:17	**Reserved** Format: MBZ
	16:6	**HUC Data Destination Base Address** Format: GraphicsAddress[16:6] Specifies the HUC Data destination base address in the L2 storage RAM. The base address is 64 byte cache aligned.
	5:0	**Reserved** Format: MBZ
5	31:17	**Reserved** Format: MBZ
	16:6	**HUC Data Length** Format: GraphicsAddress[16:6] Specifies the length in bytes of the HUC Data. The length must be in increments of 64 byte cache lines.
	5:0	**Reserved** Format: MBZ

HUC_IMEM_STATE

HUC_IMEM_STATE

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_IMEM_STATE command is used to fetch the HUC firmware from the WOPCM region and load it into the HUC 96KB L2 storage RAM. The HUC_IMEM_STATE specifies the firmware's offset in WOPCM which is a cache line aligned 32-bit offset address. The firmware, code and data segment size should always be less than 96KB. If its greater than 96KB then the firmware should be broken into smaller functions so that there is no firmware straddling case.
When the HUC_IMEM_STATE command is received, the hardware confirms that the code has been successfully authenticated by checking the VCR provided authentication successful signal. If this signal is asserted, the firmware is loaded by the HUC DMA into the 96KB L2 storage RAM. Once the firmware is loaded, the VALID IMEM LOADED bit in the HUC_STATUS2 register is asserted high in the. If the authentication signal is not asserted, the DMA aborts the HUC_IMEM_STATE command, the firmware is not loaded, and the VALID IMEM LOADED bit remains low.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 1h HUC_IMEM_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 3h — Name:

HUC_IMEM_STATE

1	31:0	**Reserved** Format: MBZ
2	31:0	**Reserved** Format: MBZ
3	31:0	**Reserved** Format: MBZ
4	31:8	**Reserved**
	7:0	**HUC Firmware Descriptor** Format: U8 This field specifies 1 of 255 firmware descriptors which describe which firmware is be loaded in the L2 storage RAM. If the firmware descriptor is set to zero, the HUC will not load the firmware.

Value	Name	Description
0		Illegal
[1,255]		Firmware Descriptor

HUC_IND_OBJ_BASE_ADDR_STATE

HUC_IND_OBJ_BASE_ADDR_STATE

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_IND_OBJ_BASE_ADDR_STATE command is used to define the indirect object base address of the stream in graphics memory. This is a frame level command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 5h HUC_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 9h — Name:
1..2	63:0	**HUC Indirect Stream In ObjectBase Address** Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the HUC_STREAM_OBJECT command for fetching (reading) the bit stream data.
3	31:0	**HUC Indirect Stream In ObjectBase Attributes** Format: MemoryAddressAttributes

HUC_IND_OBJ_BASE_ADDR_STATE		
6..7	63:0	**HUC Indirect Stream Out ObjectBase Address** Format: **SplitBaseAddress4KByteAligned** Specifies the 4K-byte aligned memory base address for the indirect data object pointed in the HUC_STREAM_OBJECT command for writing the bit stream data.
8	31:0	**HUC Indirect Stream Out ObjectBase Attributes** Format: **MemoryAddressAttributes**
9..10	63:0	**HUC Indirect Stream Out ObjectAccess Upper Bound** Format: **SplitBaseAddress4KByteAligned** This field specifies the 4K-byte aligned maximum memory address access by the indirect data object in the HUC_STREAM_OBJECT command for the output bit stream. Indirect data accessed at this address or greater will cause the HUC to stop issuing requests to the GAC. Setting this field to 0 will cause this range to be ignored by the HUC.

HUC_PIPE_MODE_SELECT

HUC_PIPE_MODE_SELECT

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_PIPE_MODE_SELECT command is responsible for general pipeline level configuration that would normally be set once for a single stream decode and would not be modified on a frame workload basis.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUP = Bh
	22:16	**Media Instruction Command** Default Value: 0h HUC_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n (Excludes Dwords 0, 1). Value: 1h — Name:
1	31:24	**Reserved**
	23:11	**Reserved** Format: MBZ
	9:5	**Reserved** Format: MBZ

HUC_PIPE_MODE_SELECT

	4	**Indirect Stream Out Enable** Format: Enable Enables the bitstream to be written out to memory immediately following the decryption of the bit stream. The memory buffer is addressed through the HuC Indirect Stream Out ObjectBase Address. **Value** / **Name** 0h — Disable Indirect Stream Out 1h — Enable Indirect Stream Out
	3:0	**Reserved** Format: MBZ
2	31:0	**Media Soft-Reset Counter (per 1000 clocks)** Format: U32 In decoder modes, this counter value specifies the number of clocks (per 1000) of GAC inactivity before a media soft-reset is applied to the HCP and HuC. If counter value is set to 0, the media soft-reset feature is disabled and no reset will occur. In encoder modes, this counter must be set to 0 to disable media soft reset. This feature is not supported for the encoder. **Value** / **Name** 0 — Disable

HUC_START

HUC_START

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 21h HUC_START Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 0h — Name:
1	31:3	**Reserved** Format: MBZ
	2:1	**Reserved**
	0	**LastStreamObject** Format: U1 Value 1 — Name: LastStreamObject — Description: Last stream object in the workload. Value 0 — Name: NotLastStreamObject — Description: Not the last stream object in the workload.

HUC_STREAM_OBJECT

HUC_STREAM_OBJECT

Source: VideoCS

Length Bias: 2

The HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_STREAM_OBJECT command is used to define the bit stream address offset to the Stream Indirect Object base Address and the length of the bit stream. The bitstream buffer the HUC operates upon is specified through indirect addressing.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 20h HUC_STREAM_OBJECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 3h — Name:

HUC_STREAM_OBJECT

1 — 31:0 — Indirect Stream In Data Length

Format:	U32

Specifies the length in bytes of the bit stream input data.

Value	Name	Description
[268435456,2147483547]	Data_Length_beyond_28_bits	This is an added valid range when the bit length is extended from 28 bits to 32 bits. This is added to support 16k x 16k picture size bitstream.

2 — 31 — Reserved

2 — 30:29 — Reserved

Format:	MBZ

2 — 28:0 — Indirect Stream In Start Address

Format:	U29

Specifies the byte-aligned graphics memory starting address of the input bit stream relative to the **HUC Indirect Stream In ObjectBase Address [31:12]**.

3 — 31:29 — Reserved

Format:	MBZ

3 — 28:0 — Indirect Stream Out Start Address

Format:	U29

Specifies the byte-aligned graphics memory starting address of the output bit stream relative to the **HUC Indirect Stream Out ObjectBase Address [31:12]**.

4 — 31:30 — Reserved

Format:	MBZ

4 — 29 — HuC Bitstream Enable

Format:	Enable

Enables the bitstream to be sent to the HuC

Value	Name
0h	Disable
1h	Enable

4 — 28:27 — DRMLengthMode

Format:	U2

Value	Name	Description
00b	Start Code Mode	Stops on a start code
01b	Length Mode	Stops after a number of bytes are reached in the length counter
10b	Reserved	
11b	Reserved	

HUC_STREAM_OBJECT

26 Stream Out

Format:	Enable

Enables the serpent encrypted stream output.

Value	Name	Description
0	Disable	Disable the serpent encrypted stream output.
1	Enable	Enable the serpent encrypted stream output.

25 Emulation Prevention Byte Removal

Format:	Enable

Value	Name	Description
0	Disable	Bypass Emulation Prevention Byte Removal.
1	Enable	Emulation prevention bytes will be removed after the start code search engine.

24 Start Code Search Engine

Format:	Enable

Value	Name	Description
0	Disable	Bypass Start Code Search Engine
1	Enable	Enables the start code search engine to stop on every third byte start code defined by **Start Code Byte [2:0]** defined in this DWord.

23:16 Start Code Byte [2]

Format:	U8

Third byte of the start code

15:8 Start Code Byte [1]

Format:	U8

Second byte of the start code

7:0 Start Code Byte [0]

Format:	U8

First byte of the start code

HUC_VIRTUAL_ADDR_STATE

HUC_VIRTUAL_ADDR_STATE

Source: VideoCS

Length Bias: 2

HUC is selected with the Media Instruction Opcode "Bh" for all HUC Commands. Each HUC command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HUC_VIRTUAL_ADDR_STATE command is used to define the 48-bit HUC Surface Base Address and HUC Surface for each region.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: Bh Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 4h HUC_VIRTUAL_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 2Fh — Name:
1..48 Regions 0 through 15 use three DWords each to describe the HUC Surface Base Address and HUC Surface, for a total of 16 regions x 3 DWords/each = 48 DWords.	95:64	**HUC Surface (VirtualAddrRegion[0-15])**
	63:0	**HUC Surface Base Address (VirtualAddrRegion[0-15])** Format: SplitBaseAddress4KByteAligned HUC surface base address for each virtual address region [0-15].

If

if - If

Source: Eulsa

Length Bias: 4

Description

An if instruction starts an if/endif or an if/else/endif block of code. It restricts execution within the conditional block to only those channels that were enabled via the predicate control.

Each if instruction must have a matching endif instruction and may have up to one matching else instruction before the matching endif.

If all channels are inactive (for the if/endif or if/else/endif block), a jump is performed to the instruction referenced by JIP. This jump must be to right after the matching else instruction when present, or otherwise to the matching endif instruction of the conditional block.

If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the 32-bit exit code <JIP> and <UIP>. If <branch_ctrl> is set, then the JIP points to the first join instruction within the if block. If <branch_ctrl> is not set, <JIP> should point to the instruction right after the matching else instruction if it exsits, otherwise <JIP> should point to the endif instruction. <UIP> should always point to the endif instruction. When a jump occurs, this value is added to IP pre-increment. In GEN instruction binary, <JIP> and <UIP> are at location <src0> & <src1> and must be of type D (signed dword integer).

Format:
[(pred)] if (exec_size JIP UIP <branch_ctrl>

Restriction

Restriction : The execution size must be the same for the if, else, and endif instructions of the same code block.

Syntax

```
[(pred)] if (exec_size) imm32 imm32 <branch_ctrl>
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn.channel[n] == 0 ) {
PcIP[n] = IP + JIP; } else { PcIP[n] = IP + 1; } } if ( PcIP != (IP + 1) ) { //
for all channels Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.

if - If		
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Illegal

illegal - Illegal

Source: Eulsa

Length Bias: 4

The Illegal Opcode Exception Enable flag in cr0.1 is normally set so the normal processing of an illegal opcode is to transfer control to the System Routine.
Instruction dispatch treats any unused 8-bit opcode (including bit 7 of the instruction, reserved for future opcode expansion) as if it is the illegal opcode.
The illegal opcode is zero because that byte value is more likely than most to be read via a wayward instruction pointer.
The illegal instruction is an instruction only in the same way that a NULL pointer in software is a pointer. Both are special values indicating invalid instances.

Format:
illegal

Restriction

Restriction : The illegal instruction takes no instruction options.

Syntax

```
illegal
```

Pseudocode

```
{ Set the Illegal Opcode Exception Status bit in cr0.1. if ( Illegal Opcode
Exception Enable is set in cr0.1 ) { Transfer control to the System Routine
(return address to AIP, IP = SIP). } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Integer Subtraction with Borrow

subb - Integer Subtraction with Borrow

Source: Eulsa

Length Bias: 4

The subb instruction performs component-wise subtraction of src0 and src1 and stores the results in dst, it also stores the borrow into acc.
If the operation produces a borrow (src0 < src1), write 0x00000001 to acc, else write 0x00000000 to acc.

Format:
[(pred)] subb[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

```
[(pred)] subb[.cmod] (exec_size) reg reg reg [(pred)] subb[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] - src1.chan[n]; acc.chan[n] = borrow(src.chan[n] - src1.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Join

join - Join

Source: Eulsa

Length Bias: 4

The join instruction makes the inactive channels active at the join IP if those channels are predicated. Any deactivated channels due to a goto instruction match the join IP are activated (qualified with predicates at join). If no IP is matched at this join, the program goes to the next IP with the active channels which followed the program path up to the join instruction. If no active channels are present after executing the join instruction, the program jumps to the offset specified by JIP instead of next IP.

The join instruction is used in conjunction with a goto instruction. The join activates channels that are deactivated by the goto instruction. See the goto instruction for the deactivation rules.

The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence.

The following table describes the 32-bit JIP. In GEN binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand and is a signed 32-bit number. This value is added to IP pre-increment.

If SPF is ON, none of the PcIP are updated.

Format:

[(pred)] join (exec_size) JIP

Programming Notes

An index of 0 does nothing, continuing execution with the next instruction.

An index of -16 (if the jmpi instruction is in native format) or -8 (if the jmpi instruction is in compact format) is an infinite loop on the jmpi instruction.

Restriction

Restriction : The {NoMask} instruction option must be specified.

Restriction : The index data type must be D (Signed DWord Integer).

Syntax

```
[(pred)] join (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if (WrEn.chan[n] ) { // for
the predicated channels and the remaining channels PcIP[n] = IP + 1; } } if (
PcIP != (IP + 1) ) { // for all channels when no channels are activated and no
other active channels Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier

join - Join

Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: **SrcType**
	90:89	**Src1.RegFile** Format: **RegFile**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Jump Indexed

jmpi - Jump Indexed

Source: Eulsa

Length Bias: 4

Description

The jmpi instruction redirects program execution to an index offset relative to the post-incremented instruction pointer. The index is a signed integer value, with positive or zero integers for forward jumps, and negative integers for backward jumps.

Note: Unlike other flow control instructions, the offset used by jmpi is relative to the incremented instruction pointer rather than the IP value for the instruction itself.

In GEN binary, index is at location src1. The ip register must be put (for example, by the assembler) at the dst and src0 locations.

Predication is allowed to provide conditional jump with a scalar condition. As the execution size is 1, the first channel of PMASK (flags post prediction control and negate) is used to determine whether the jump is taken or not. If the condition is false, the jump is not taken and execution continues with the next instruction.

Format:

[(pred)] jmpi (1) index {NoMask}

Programming Notes

An index of 0 does nothing, continuing execution with the next instruction.

An index of -16 (if the jmpi instruction is in native format) or -8 (if the jmpi instruction is in compact format) is an infinite loop on the jmpi instruction.

Restriction

Restriction : The execution size must be 1.

Restriction : The {NoMask} instruction option must be specified.

Restriction : The index data type must be D (Signed DWord Integer).

Restriction : QtrCtrl must not be used for jmpi instruction.

Syntax

```
[(pred)] jmpi (1) reg32 {NoMask} [(pred)] jmpi (1) imm32 {NoMask}
```

Pseudocode

```
Evaluate(WrEn); if ( WrEn != 0 ) { Jump(IP + 1 + index ); // IP + 1 is a
pseudocode idiom for the IP of the following instruction. }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types

jmpi - Jump Indexed

D

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: **SrcType**
	90:89	**Src1.RegFile** Format: **RegFile**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Leading Zero Detection

lzd - Leading Zero Detection

Source: Eulsa

Length Bias: 4

The lzd instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is zero, store 32 in dst.

Format:
[(pred)] lzd[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] lzd[.cmod] (exec_size) reg reg [(pred)] lzd[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD udScalar = src0.chan[n]; UD cnt = 0; while ( (udScalar & (1 « 31)) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar « 1; } dst.chan[n] = cnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
D, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG**
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_IMM32**
	63:32	**Operand Controls** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Line

line - Line

Source: Eulsa

Length Bias: 4

The line instruction computes a component-wise line equation (v = p * u + q where u, v are vectors and p, q are scalars) of src0 and src1 and stores the results in dst. src1 is the input vector u. src0 provides input scalars p and q, where p is the scalar value based on the region description of src0 and q is the scalar value implied from src0 region. Specifically, q is the fourth component of the 4-tuple (128-bit aligned) that p belongs to.

Format:
[(pred)] line[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : This is a specialized instruction that only supports an execution size (ExecSize) of 8 or 16.

Restriction : The src0 region must be a replicated scalar (with HorzStride == VertStride == 0).

Restriction : src0 must specify .0 or .4 as the subregister number, corresponding to a subregister byte offset of 0 or 16.

Restriction : Source operands cannot be accumulators.

Syntax

```
[(pred)] line[.cmod] (exec_size) reg reg reg [(pred)] line[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { dwP = src0.RegNum.SubRegNum[bits4:2]; // A DWord-aligned scalar. dwQ = src0.RegNum.(SubRegNum[bit4] | 0x8); // Fourth component. if ( WrEn.chan[n] ) { dst.chan[n] = dwP * src1.chan[n] + dwQ; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM

line - Line		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Linear Interpolation

lrp - Linear Interpolation

Source: Eulsa

Length Bias: 4

The lrp instruction takes component-wise multiplication of src0 and src1, and adds the result to the component-wise multiplication of src2 and (1 - src0), and then stores the final results in dst.

Format:
[(pred)] lrp[.cmod] (exec_size) dst src0 src1 src2

Restriction

Restriction : The vertical stride (VertStride) is overloaded to 4 in HW for 3-source instructions.

Restriction : The overflow conditional modifier (.o) is not allowed.

Restriction : No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.

Restriction : All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] lrp[.cmod] (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) {
dst.chan[n] = src1.chan[n] * src0.chan[n] + src2.chan[n] * (1.0 - src0.chan[n]);
} }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

lrp - Linear Interpolation

	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group
	48	**Reserved** Format: MBZ
	47	**NibCtrl** Format: NibCtrl
	46	**Reserved** Format: MBZ
	45:44	**Destination Data Type** This field contains the data type for the destination

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

	48:42	**Reserved** Format: MBZ
	43:42	**Source Data Type** This field contains the data type for all three sources

Value	Name
00b	Single Precision Float
01b	DWord

lrp - Linear Interpolation

		10b: Unsigned DWord 11b: Double Precision Float
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: **SrcMod**
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	34	**Flag Register Number** This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Destination Register File** Value: 0 — Name: GRF Value: 1 — Name: MRF
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

LO8DS Render Target Write MSD

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

17:14 Message Type

Default Value:	0Ch
Format:	Opcode

Render Target Write message

13 Per-Sample PS Enable

Format:	Enable

If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.

Programming Notes

This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
This bit MUST be 0 for Dual Source Messages.

12 Last Render Target Select

Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set.
In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 Slot Group Select

Format:	MDC_RT_SGS

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

10:8 Render Target Message Subtype

Default Value:	2h
Format:	Opcode

SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask.

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

		Programming Notes The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Logic And

and - Logic And

Source: Eulsa

Length Bias: 4

The and instruction performs component-wise logic AND operation between src0 and src1 and stores the results in dst.
Register source operands can use source modifiers:
Any source modifier is numeric, optionally changing a source value s to -s, abs(s), or -abs(s) before the AND operation.
Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a AND (NOT b) to be calculated with one instruction.
This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Format:
Source modifier is not allowed if source is an accumulator.

Restriction

Restriction : Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] and[.cmod] (exec_size) reg reg reg [(pred)] and[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] & src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

<table>
<tr><th colspan="3">and - Logic And</th></tr>
<tr><td></td><td>127:64</td><td>ImmSource<table><tr><td>Exists If:</td><td>([ImmSource][Src1.RegFile]=='IMM')</td></tr><tr><td>Format:</td><td>EU_INSTRUCTION_SOURCES_REG_IMM</td></tr></table></td></tr>
<tr><td></td><td>63:32</td><td>Operand Controls<table><tr><td>Format:</td><td>EU_INSTRUCTION_OPERAND_CONTROLS</td></tr></table></td></tr>
<tr><td></td><td>31:0</td><td>Header<table><tr><td>Format:</td><td>EU_INSTRUCTION_HEADER</td></tr></table></td></tr>
</table>

Logic Not

not - Logic Not

Source: Eulsa

Length Bias: 4

Description

The not instruction performs logical NOT operation (or one's complement) of src0 and storing the results in dst.
This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

A register source operand can use a source modifier:
Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). Such a source modifier is not particularly useful with the not instruction, as it changes the effect of not to just copying bits.

Format:
[(pred)] not[.cmod] (exec_size) dst src0

Restriction

Restriction : Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] not[.cmod] (exec_size) reg reg [(pred)] not[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) {
dst.chan[n] = ~ src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG

not - Logic Not		
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Logic Or

or - Logic Or

Source: Eulsa

Length Bias: 4

Description

The or instruction performs component-wise logic OR operation between src0 and src1 and stores the results in dst.
This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Register source operands can use source modifiers:
Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a OR (NOT b) to be calculated with one instruction.

Format:
[(pred)] or[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] or[.cmod] (exec_size) reg reg reg [(pred)] or[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] | src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_REG**

or - Logic Or

	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Logic Xor

xor - Logic Xor

Source: Eulsa

Length Bias: 4

Description

The xor instruction performs component-wise logic XOR operation between src0 and src1 and stores the results in dst.
This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Register source operands can use source modifiers:
Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a XOR (NOT b) to be calculated with one instruction.

Format:
[(pred)] xor[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] xor[.cmod] (exec_size) reg reg reg [(pred)] xor[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] ^ src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

xor - Logic Xor		
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

MEDIA_CURBE_LOAD

MEDIA_CURBE_LOAD

Source: RenderCS

Length Bias: 2

Workaround : See "GPGPU Command Workarounds" section for additional programming constraints for this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_CURBE_LOAD Format: OpCode
	23:16	**SubOpcode** Default Value: 1h MEDIA_CURBE_LOAD SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value | Name | Description 2h | DWORD_COUNT_n **[Default]** | Excludes DWord (0,1)
1	31:0	**Reserved** Format: MBZ
2	31:17	**Reserved** Format: MBZ
	16:0	**CURBE Total Data Length** Format: U17 In Bytes Description This field provides the length in bytes of the CURBE data. This field must have the same alignment as the Curbe Object Data Start Address.As the CURBE data are sent directly to ROB, range is limited to CURBE Allocation Size. This field must be 64-byte aligned.

MEDIA_CURBE_LOAD

3	31:0	**CURBE Data Start Address** Format: DynamicStateOffset[31:0] CURBE **Description** Specifies the 64-byte aligned address of the CURBE data. This pointer is relative to the **Dynamics Base Address**. **Value**: [0, FFFFFFFFh] **Name**:

MEDIA_INTERFACE_DESCRIPTOR_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Source: RenderCS

Length Bias: 2

A Media_State_Flush should be used before this command to ensure that the temporary Interface Descriptor storage is cleared.

Workaround : See "GPGPU Command Workarounds" section for additional programming constraints for this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_INTERFACE_DESCRIPTOR_LOAD Format: OpCode
	23:16	**SubOpcode** Default Value: 2h MEDIA_INTERFACE_DESCRIPTOR_LOAD SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 2h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:0	**Reserved** Format: MBZ
2	31:17	**Reserved** Format: MBZ
	16:0	**Interface Descriptor Total Length** Format: U17 In bytes This field provides the length in bytes of the Interface Descriptor data. This field must have the same alignment as the Interface Descriptor Data Start Address.It must be DQWord (32-byte) aligned. As the Interface Descriptor data are sent directly to ROB, range is limited to CURBE Allocation Size.

MEDIA_INTERFACE_DESCRIPTOR_LOAD

		Value / **Name** [32,2048] / [1,64] interface descriptor entries
3	31:0	**Interface Descriptor Data Start Address** Format: DynamicStateOffset[31:0]INTERFACE_DESCRIPTOR_DATA **Description** This bit specifies the 64-byte aligned address of the Interface Descriptor data. This pointer is relative to the Dynamics Base Address. **Value** / **Name** [0, FFFFFFFFh] /

MEDIA_OBJECT

MEDIA_OBJECT

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT Format: OpCode
	23:16	**Media Command Sub-Opcode** Default Value: 0h MEDIA_OBJECT SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 4h DWORD_COUNT_n Format: =n Total Length - 2 Excludes DWords 0,1 **Generic Mode:** DWord Length = N+4, where N is in the range of [0,504]. The maximum is 504 DW (equivalent to 63 8-DW registers). When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length N and indirect data length rounded up to 8-DW aligned individually). The minimal inline data length is 0.
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.

MEDIA_OBJECT

2 — 31 — Children Present

Format:	Enable

Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads.

If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched.

If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread.

In order avoid deadlock, such dereference must be issued once and only once for each URB handle.

30:27 — Reserved

Format:	MBZ

26:25 — Reserved

24 — Thread Synchronization

This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message.

Value	Name
0	No thread synchronization
1	Thread dispatch is synchronized by the 'spawn root thread' message

23 — Reserved

Format:	MBZ

22 — Force Destination

If set, bits 20:17 are used to determine the destination of this dispatch, if clear the destination will be chosen based on load.

21 — Use Scoreboard

This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard.

Value	Name
0	Not using scoreboard
1	Using scoreboard

20:19 — Slice Destination Select

This bit along with the subslice destination select determines the slice that this thread must be sent to. Ignored if **Force Destination** = 0, or if product only has 1 slice.

Value	Name	Description
00b	Slice 0	
01b	Slice 1	Cannot be used in products without a Slice 1.
10b	Slice 2	Cannot be used in products without a Slice 2.

MEDIA_OBJECT

		11b Reserved
	18:17	**SubSlice Destination Select** This field selects the SubSlice that this thread must be sent to. Ignored if **Force Destination** = 0 Value: 11b — Name: Subslice 3 Value: 10b — Name: SubSlice 2 Value: 01b — Name: SubSlice 1 Value: 00b — Name: SubSlice 0
	16:0	**Indirect Data Length** Format: U17 In bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to (URB Entry Allocation Size)*(Number of URB Entries) in the MEDIA_VFE_STATE command. When both inline and indirect data are fetched for this command, enough room should be allowed for at least 1 copy of the indirect and 1 copy of the inline data in the URB.
3	31:0	**Indirect Data Start Address** Format: GraphicsAddress[31:0] **Description** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. This field specifies the 64-byte aligned address of the indirect data. Value: [0,512MB] — Name: **Programming Notes** Bits 31:29 MBZ
4	31:25	**Reserved** Format: MBZ
	24:16	**Scoredboard Y** Format: U9 This field provides the Y term of the scoreboard value of the current thread.

MEDIA_OBJECT

	15:9	**Reserved** Format: MBZ
	8:0	**Scoreboard X** Format: U9 This field provides the X term of the scoreboard value of the current thread.
5	31:20	**Reserved** Format: MBZ
	19:16	**Scoreboard Color** Format: U4 This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control.
	15:8	**Reserved** Format: MBZ
	7:0	**Scoreboard Mask** Format: Boolean Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE command. **Bit n (for n = 0…7):** Scoreboard n is dependent, where bit 0 maps to n = 0.
6..n	31:0	**Inline Data** Generic Mode: The format of this data is specified by software. Hardware does not interpret this data; it merely passes it to the kernel for processing. The total size for the inline data and indirect data must not exceed 112 registers.

MEDIA_OBJECT_GRPID

MEDIA_OBJECT_GRPID

Source: RenderCS

Length Bias: 2

The MEDIA_OBJECT_GRPID command is a variation of MEDIA_OBJECT which includes a group id which is used to allocate and track Barriers and Shared Local Memory. The Interface Descriptor is used to specify how much SLM is needed and how many threads will be reporting to the Barrier. All MEDIA_OBJECT_GRPIDs with the same group id should have the same interface descriptor and be dispatched to the same Tslice – the dispatcher will ensure this if Force Destination = 0, but software must ensure this if Force Destination = 1. Software should also ensure that all the threads needed for the Barrier will fit into a Tslice, or the Barrier will never be satisfied. Either SLM or a barrier must be used with MEDIA_OBJECT_GRPID, if neither is needed then a MEDIA_OBJECT must be used instead.

MEDIA_OBJECT_GRPID supports the GPGPU version of payload delivery – either indirect or CURBE can be split between the threads in a group (per-thread payload), as well as a section which is sent to all threads (cross-thread payload). See the GPGPU payload section. For indirect, the same pointer must be sent with all the commands associated with the thread group for payload splitting to work properly. Inline data is not split, but the payload attached to each command is sent with that thread. Only one of inline, indirect, or CURBE is allowed, but at least one form of payload must be sent.

MEDIA_STATE_FLUSH with the watermark bit must be placed between groups created by MEDIA_OBJECT_GRPID. The Interface Descriptor associated with the watermark must match the Interface Descriptor used for the following group.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_GRPID Format: OpCode
	23:16	**Media Command Sub-Opcode** Default Value: 6h MEDIA_OBJECT_GRPID SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 5h DWORD_COUNT_n Format: =n Total Length - 2 Excludes DWords 0,1 **Generic Mode:** DWord Length = N+5, where N is in the range of [0,504]. The maximum is 504

MEDIA_OBJECT_GRPID

		DW (equivalent to 63 8-DW registers). When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length N and indirect data length rounded up to 8-DW aligned individually). The minimal inline data length is 0.
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. **Value** / **Name** [0,30]
2	31:25	**Reserved** Format: MBZ
	24	**Reserved**
	23	**End of Thread Group** This bit indicates that this dispatch is the last for the current thread group.
	22	**Force Destination** If set, bits 20:17 are used to determine the destination of this dispatch, if clear the destination will be chosen based on load.
	21	**Use Scoreboard** This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard. **Value** / **Name** 0 / Not using scoreboard 1 / Using scoreboard
	20:19	**Slice Destination Select** This bit along with the SubSlice destination select determines the slice that this thread must be sent to. Ignored if **Force Destination** = 0, or if product only has 1 slice. **Value** / **Name** / **Description** 00b / Slice 0 / 01b / Slice 1 / Cannot be used in products without a Slice 1. 10b / Slice 2 / Cannot be used in products without a Slice 2. 11b / Reserved /

MEDIA_OBJECT_GRPID

	18:17	**SubSlice Destination Select** This field selects the SubSlice that this thread must be sent to. Ignored if **Force Destination** = 0 **Value** / **Name** 11b – Subslice3 10b – SubSlice 2 01b – SubSlice 1 00b – SubSlice 0
	16:0	**Indirect Data Length** Format: U17 In bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to(URB Entry Allocation Size)*(Number of URB Entries) in the MEDIA_VFE_STATE command.
3	31:0	**Indirect Data Start Address** Format: GraphicsAddress[31:0] **Description** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. It is the 64-byte aligned address of the indirect data. **Value** / **Name** / **Description** [0-512MB] / / Bits 31:29 MBZ
4	31:25	**Reserved** Format: MBZ
	24:16	**Scoreboard Y** Format: U9 This field provides the Y term of the scoreboard value of the current thread.
	15:9	**Reserved** Format: MBZ
	8:0	**Scoreboard X** Format: U9 This field provides the X term of the scoreboard value of the current thread.

MEDIA_OBJECT_GRPID

5	31:20	**Reserved** Format: MBZ
	19:16	**Scoreboard Color** Format: U4 This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control.
	15:8	**Reserved** Format: MBZ
	7:0	**Scoreboard Mask** Format: Boolean Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE command. **Bit n (for n = 0…7):** Scoreboard n is dependent, where bit 0 maps to n = 0.
6	31:0	**GroupID** A unique identifying number which describes the threads which share a barrier and/or SLM. Reuse of numbers is allowed as long as the old group is not currently running.
7..n	31:0	**Inline Data** The format of this data is specified by software. Hardware does not interpret this data; it merely passes it to the kernel for processing. The total size for the inline data and indirect data must not exceed 112 registers.

MEDIA_OBJECT_PRT

MEDIA_OBJECT_PRT

Source: RenderCS

Length Bias: 2

command is for generating Persistent Root Thread for the media pipeline. It only supports loading of inline data but not indirect data.
This command should be used for a root thread that might have to be present in the system for a period longer than the certain minimal context-switch interrupt latency. It has to honor the context interrupt signal to terminate upon request. It should also handle replay from theinterrupted point upon context restore (the same thread being dispatched more than once). In contrary, if a thread is not a Persistent Root Thread, if dispatched, it must run to completion.
The command can be used in all VFE modes, except VLD mode.

For simplification, _PRT command has a fixed size of 16 DWORD

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_PRT Format: OpCode
	23:16	**SubOpcode** Default Value: 2h MEDIA_OBJECT_PRT SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Note: Regardless of the mode, inline data must be present in this command. The command size must fit within 16 dwords. Value: 0Eh — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.

MEDIA_OBJECT_PRT

2	31	**Children Present** Format: Enable Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. In order avoid deadlock, such de-reference must be issued once and only once for each URB handle.
	30:24	**Reserved** Format: MBZ
	23	**PRT_Fence Needed** Format: Enable This field specifies that a PRT_Fence is generated after dispatching the thread associated with this MEDIA_OBJECT_PRT. The PRT_Fence prevents additional threads following this persistent root thread until a thread spawn message is sent. The PRT_Fence is generated on first dispatch of the persistent root, as well as on re-dispatches of the persistent root after context restore.
	22	**PRT_FenceType** This field specifies the type of fence the PRT thread uses. If this field is set to 0, the fence is set at the end of the root thread queue. It will block the dispatch of the next root thread, but allowed these root threads to be populated through VFE to the root thread queue in TS. If this field is set to 1, the fence is set at the entry of VFE, similar to the fence set by the MEDIA_STATE_FLUSH command. No more command can go into the media pipe until a thread spawn message is sent (by the PRT). This field is only valid when PRT_Fence Needed is set to 1. Otherwise, it is ignored by hardware. **Value** / **Name** / **Description** 0h / Root thread queue / Root thread queue fence 1h / VFE state flush / VFE state flush fence
	21:0	**Reserved** Format: MBZ
3	31:0	**Reserved** Format: MBZ
4..15	31:0	**Inline Data** Format: U32

MEDIA_OBJECT_WALKER

MEDIA_OBJECT_WALKER

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_WALKER Format: OpCode
	23:16	**SubOpcode** Default Value: 03h MEDIA_OBJECT_WALKER SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 0Fh DWORD_COUNT_n Format: =n Total Length - 2 **Note:** If this field is greater than 15, it indicates that inline data is present. If present, inline data is common for all threads generated from this command, If this field is 15, it indicates that inline data is not present. It should be noted that unlike other media object command, inline data is optional for this command.
1	31:8	**Reserved**
	7:6	**Reserved** Format: Reserved
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31:25	**Reserved** Format: MBZ
	24	**Thread Synchronization** This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message.

MEDIA_OBJECT_WALKER

Value	Name
0	No thread synchronization
1	Thread dispatch is synchronized by the 'spawn root thread' message

23:22 **Masked Dispatch**

Format:	U2

Enable the masking of the dispatch of individual threads based on a bitmask read from CURBE, and specifies the pitch of the CURBE surface.
If enabled, CURBE will not be used for thread payload.

Value	Name	Description
00b		Masked Dispatch Disabled
01b		Masked Dispatch with 128-bit pitch in CURBE
10b		Masked Dispatch with 256-bit pitch in CURBE
11b		Masked Dispatch with 512-bit pitch in CURBE

21 **Use Scoreboard**

This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard.

Value	Name
0	Not using scoreboard
1	Using scoreboard

20:17 **Reserved**

Format:	MBZ

16:0 **Indirect Data Length**

Format:	U17 in bytes

This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, the range is limited to the (URB Entry Allocation Size)*(Number of URB Entries) specified in the MEDIA_VFE_STATE command.

3 31:0 **Indirect Data Start Address**

This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address.**
Hardware ignores this field if indirect data is not present.
Alignment of this address depends on the mode of operation.

It is the 64-byte aligned address of the indirect data

Value	Name	Description
[0 - 512MB]		(Bits 31:29 MBZ)

MEDIA_OBJECT_WALKER

4 — 31:0 — Reserved

Format:	MBZ

5 — 31:8 — Group ID Loop Select

This bit field chooses which of the nested loops of the walker are used to identify threads which share a group id and therefore a shared barrier and SLM. The programmer must ensure that each group will fit into a single subslice. When barriers are enabled every group must have the same number of threads matching the number specified in the Interface Descriptor.

Value	Name	Description
0	No_Groups	Groups are not created, barriers and SLM are not allocated
1	Color_Groups	Each complete iteration of the Color loop defines a group, the group id is the concatenation of the Outer global, Inner global, Outer local, Mid local and Inner local loop execution counts.
2	InnerLocal_Groups	Each complete iteration of the Inner local loop and Color loop defines a group, the group id is the concatenation of the Outer global loop to the Mid local loop execution counts.
3	MidLocal_Groups	Each complete iteration of the Mid local loop and lower loops defines a group, the group id is the concatenation of the Outer global loop to the Outer local loop execution counts.
4	OuterLocal_Groups	Each complete iteration of the Outer local loop and lower loops defines a group, the group id is the concatenation of the Outer global loop and the Inner global loop execution counts.
5	InnerGlobal_Groups	Each complete iteration of the Inner global loop and lower loops defines a group, the group id is the Outer global loop execution count.

5 — 7:0 — Scoreboard Mask

Format:	Boolean

Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. All threads generated by this walker command share the same dynamic mask.
Bit n (for n = 0...7): Scoreboard n is dependent, where bit 0 maps to n = 0.

6 — 31:28 — Reserved

6 — 27:24 — Color Count Minus One

Format:	U4

This field specifies the number of repeat of the inner most loop of the walker. Each repeated walk position is assigned with an incremental Color number. The Color number together with the X and Y position of the thread is used for dependency scoreboard control.
Usage Example: This allows multiple sets of dependency threads to be dispatched.

6 — 23:21 — Reserved

Format:	MBZ

MEDIA_OBJECT_WALKER

	20:16	**Middle Loop Extra Steps** Format: U5
	15:14	**Reserved** Format: MBZ
	13:12	**Local Mid-Loop Unit Y** Format: S1
	11:10	**Reserved** Format: MBZ
	9:8	**Mid-Loop Unit X** Format: S1
	7:0	**Reserved** Format: MBZ
7	31:28	**Reserved** Format: MBZ
	27:16	**Global Loop Exec Count** Format: U12
	15:12	**Reserved** Format: MBZ
	11:0	**Local Loop Exec Count** Format: U12
8	31:27	**Reserved** Format: MBZ
	26:16	**Block Resolution Y** Format: U11 Vertical resolution of the local loop.
	15:11	**Reserved** Format: MBZ
	10:0	**Block Resolution X** Format: U11 Horizontal resolution of the local loop.
9	31:27	**Reserved** Format: MBZ
	26:16	**Local Start Y** Format: U11 Starting vertical position of the local loop.

MEDIA_OBJECT_WALKER		
	15:11	**Reserved** Format: MBZ
	10:0	**Local Start X** Format: U11 Starting horizontal position of the local loop.
10	31:0	**Reserved** Format: MBZ
11	31:28	**Reserved** Format: MBZ
	27:16	**Local Outer Loop Stride Y** Format: S11 Vertical stride of the local outer loop, in 2's complement.
	15:12	**Reserved** Format: MBZ
	11:0	**Local Outer Loop Stride X** Format: S11 Horizontal stride of the local outer loop, in 2's complement.
12	31:28	**Reserved** Format: MBZ
	27:16	**Local Inner Loop Unit Y** Format: S11 Vertical stride of the local inner loop, in 2's complement.
	15:12	**Reserved** Format: MBZ
	11:0	**Local Inner Loop Unit X** Format: S11 Horizontal stride of the local inner loop, in 2's complement.
13	31:27	**Reserved** Format: MBZ
	26:16	**Global Resolution Y** Format: U11 Vertical resolution of the global loop.

MEDIA_OBJECT_WALKER

	15:11	**Reserved** Format: MBZ
	10:0	**Global Resolution X** Format: U11 Horizontal resolution of the global loop.
14	31:28	**Reserved** Format: MBZ
	27:16	**Global Start Y** Format: S11 Starting vertical location of the global loop, in 2's complement.
	15:12	**Reserved** Format: MBZ
	11:0	**Global Start X** Format: S11 Starting horizontal location of the global loop, in 2's complement.
15	31:28	**Reserved** Format: MBZ
	27:16	**Global Outer Loop Stride Y** Format: S11 Vertical stride of the global outer loop, in 2's complement.
	15:12	**Reserved** Format: MBZ
	11:0	**Global Outer Loop Stride X** Format: S11 Horizontal stride of the global outer loop, in 2's complement.
16	31:28	**Reserved** Format: MBZ
	27:16	**Global Inner Loop Unit Y** Format: S11 Vertical stride of the global inner loop, in 2's complement.
	15:12	**Reserved** Format: MBZ

MEDIA_OBJECT_WALKER		
	11:0	**Global Inner Loop Unit X** Format: S11 Horizontal stride of the global inner loop, in 2's complement.
17..n	31:0	**Inline Data**

MEDIA_STATE_FLUSH

MEDIA_STATE_FLUSH

Source: RenderCS

Length Bias: 2

This command updates the Message Gateway state. In particular, it updates the state for a selected Interface Descriptor.

This command can be considered same as a MI_Flush except that only media parser will get flushed instead of the entire 3D/media render pipeline. The command should be programmed prior to new Media state, curbe and/or interface descriptor commands when switching to a new context or programming new state for the same context. With this command, pipelined state change is allowed for the media pipe.

Be cautious when using this command when child_present flag in the media state is enabled. This is because that CURBE state as well as Interface Descriptor state are shared between root threads and child threads. Changing these states while child threads are generated on the fly may cause unexpected behavior. Combining with MI_ARB_ON/OFF command, it is possible to support interruptability with the following command sequence where interrupt may be allowed only when MI_ARB_ON_OFF is ON:

```
MEDIA_STATE_FLUSH
 VFE_STATE // VFE will hold CS if watermark isn't met
 MI_ARB_OFF // There must be at least one VFE command before this one
 MEDIA_OBJECT ... MI_ARB_ON
```

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_STATE_FLUSH Format: OpCode
	23:16	**SubOpcode** Default Value: 4h MEDIA_STATE_FLUSH SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 0h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:9	**Reserved** Format: MBZ

MEDIA_STATE_FLUSH

	8	**Reserved**
	7	**Flush to GO** Format: Enable This bit indicates that the write data out of this thread group should be flushed to the point where it is visible to following commands.
	6	**Watermark Required** This is a single bit specifying if the MEDIA_STATE_FLUSH should stall further commands until there is enough room in a half-slice for the following thread group. The characteristics of the thread group are specified in the Interface Descriptor Offset. If set, the MEDIA_STATE_FLUSH stalls CS until there are enough threads in a half-slice, and enough SLM available in the same half-slice, and a free barrier if one is required. An Interface Descriptors can be updated after a Watermarked MEDIA_STATE_FLUSH only if it has not been used in the current context. Reusing an interface desciptor requires that this bit is clear to ensure the ID cache is reloaded. If clear, the MEDIA_STATE_FLUSH stalls CS until the TDL has dispatched the last thread, allowing the CURBE and Interface Descriptors to be updated by following commands. **Programming Notes** The Interface Descriptor Offset used for the flush must be the same as that used for the GPGPU_OBJECTs. GPGPU_WALKER automatically checks the Watermark conditions before starting a thread, so this bit should not be set before GPGPU_WALKER.
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which describes what resources are required to meet the watermark.

MEDIA_VFE_STATE

MEDIA_VFE_STATE

Source: RenderCS

Length Bias: 2

A stalling PIPE_CONTROL is required before MEDIA_VFE_STATE unless the only bits that are changed are scoreboard related: Scoreboard Enable, Scoreboard Type, Scoreboard Mask, Scoreboard * Delta. For these scoreboard related states, a MEDIA_STATE_FLUSH is sufficient.

- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_VFE_STATE Format: OpCode
	23:16	**SubOpcode** Default Value: 0h MEDIA_VFE_STATE SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 07h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[31:10] Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the **General State Base Address.**
	9:8	**Reserved** Format: MBZ

MEDIA_VFE_STATE

7:4 Stack Size

Value	Name	Description
[0,11]		indicating [1KBytes, 2MBytes]

Programming Notes
Since the stack uses the upper portion of the scratch space, **Stack Size =< Per Thread Scratch Space**

3:0 Per Thread Scratch Space

Format:	U4

Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the maximum threads in the device each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space.

Value	Name	Description
[0,11]		indicating [1k bytes, 2 Mbytes]: 0=1k, 1=2k, 2=4k, 3=8k ... 11=2M

2 31:16 Reserved

Format:	MBZ

15:0 Scratch Space Base Pointer High

Format:	GeneralStateOffset[47:32]

This field specifies the high 16 bits of starting address of the Scratch Space Base Pointer

3 31:16 Maximum Number of Threads

Format:	U16-1 representing thread count

Range: [0, n-1] where n = (# EUs) * (# threads/EU).
See *Graphics Processing Engine* for listing of #EUs and #threads in each device.

Specifies the maximum number of simultaneous root threads allowed to be active. Used to avoid potential deadlock. If child threads are not planning on being used then this field can be set to its maximum value and there will be no thread limit beyond what is currently available in the system; the maximum value can include threads in slices that have been shut down for power reasons. For GPGPU threads the maximum value must be used.

Programming Notes
MSB will be zero due to the range limit below.

15:8 Number of URB Entries

Format:	U8

Specifies the number of URB entries that are used by the unit.

Value	Name	Description
[1,128]		[1,128] Entries

MEDIA_VFE_STATE

		Programming Notes Please note that 0 is not allowed for this field.
	7	**Reset Gateway Timer** This field controls the reset of the timestamp counter maintained in Message Gateway. Value: 0h — Maintaining the existing timestamp state Value: 1h — Resetting relative timer and latching the global timestamp
	6	**Bypass Gateway Control** This field configures Gateway to use a simple message protocol. Value: 0h — Maintaining OpenGateway/ForwardMsg/CloseGateway protocol (legacy mode) Value: 1h — Bypassing OpenGateway/CloseGateway protocol
	6	**Reserved** Format: MBZ
	5:3	**Reserved** Format: MBZ
	2	**Reserved** Format: MBZ
	1:0	**Reserved**
4	31:8	**Reserved**
	7:4	**Reserved** Format: MBZ
	3:2	**Reserved** Format: MBZ
	1:0	**Slice Disable** This field disables dispatch to slices and subslices for Media and GPGPU applications. It is used to limit the amount of scratch space that needs to be allocated for a context. If a particular configuration doesn't have slice or subslice then there is no impact to disabling it. Value: 00b — Name: All Subslices Enabled — Description: All subslices are enabled. Value: 01b — Name: Only Slice 0 Enabled — Description: Slice 2 and 1 are disabled, only Slice 0 with all subslices is enabled. Value: 11b — Name: Only Slice 0 Subslice 0 Enabled — Description: Slice 2 and 1 are disabled, only Slice 0 with only subslice 0 enabled.

MEDIA_VFE_STATE

5	31:16	**URB Entry Allocation Size** Format: U16-1 **Description** Specifies the length of each URB entry used by the unit, in 256-bit register increments. ROB address for URB starts after CURBE Allocated region. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + Number of Interface Descriptors) must be less than (number of bytes allocated for the URB in L3CNTLREG / 32 bytes per entry). Note: Number of Interface Descriptors is 64. If SLM is enabled for GPGPU work then the number of available entries will be 1/2 the maximum URB entries. **Programming Notes** When Inline data is used with MEDIA_OBJECT or MEDIA_OBJECT_WALKER, then the URB entry allocation size must match the Inline data size. If Indirect data is being used with MEDIA_OBJECT or GPGPU_WALKER then the allocation size must be sufficient for the Indirect data. If both Inline and Indirect are being used, then the allocation size must match the sum of the Inline and Indirect.
	15:0	**CURBE Allocation Size** Format: U16-1 **Description** Specifies the total length allocated for CURBE, in 256-bit register increments. ROB address for CURBE starts at address 64. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + Interface Descriptor Entries) must be less than or equal to the number of entries in the URB as described in **Configurations**. Interface Descriptor Entries is 64 If SLM is enabled for GPGPU work then the number of available entries will be ½ the maximum URB entries. **Programming Notes** CURBE Allocation Size should be 0 for GPGPU workloads that uses indirect instead of CURBE.
6	31	**Scoreboard Enable** Format: Enable This field enables and disables the hardware scoreboard in the Media Pipeline. If this field is cleared, hardware ignores the following scoreboard state fields.

Value	Name
0h	Scoreboard disabled
1h	Scoreboard enabled

MEDIA_VFE_STATE

30 — Scoreboard Type

This field selects the type of scoreboard in use.

Value	Name
0h	Stalling Scoreboard
1h	Non-Stalling Scoreboard

29:16 — Reserved

Format:	MBZ

15:8 — Reserved

Format:	MBZ

7:0 — Scoreboard Mask

Format:	Enable(8)

Each bit indicates the corresponding dependency scoreboard is enabled. The scoreboard is based on the relative (X, Y) distance from the current threads' (X, Y) position. **Bit n (for n = 0…7):** Score n is enabled.

7 — 31:28 — Scoreboard 3 Delta Y

Format:	S3

Relative vertical distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.

Programming Notes
MBZ if scoreboard is disabled.

27:24 — Scoreboard 3 Delta X

Format:	S3

Relative horizontal distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.

Programming Notes
MBZ if scoreboard is disabled.

23:20 — Scoreboard 2 Delta Y

Format:	S3

Relative vertical distance of the dependent instance assigned to scoreboard 2, in the form of 2's compliment.

Programming Notes
MBZ if scoreboard is disabled.

19:16 — Scoreboard 2 Delta X

Format:	S3

Relative horizontal distance of the dependent instance assigned to scoreboard 2, in the form of 2's compliment.

Programming Notes
MBZ if scoreboard is disabled.

MEDIA_VFE_STATE

	15:12	**Scoreboard 1 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 1, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	11:8	**Scoreboard 1 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 1, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	7:4	**Scoreboard 0 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 0, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	3:0	**Scoreboard 0 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 0, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
8	31:28	**Scoreboard 7 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 7, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	27:24	**Scoreboard 7 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 7, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.

MEDIA_VFE_STATE

	23:20	**Scoreboard 6 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 6, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	19:16	**Scoreboard 6 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 6, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	15:12	**Scoreboard 5 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 5, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	11:8	**Scoreboard 5 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 5, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	7:4	**Scoreboard 4 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 4, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	3:0	**Scoreboard 4 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 4, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.

Media Block Read MSD

MSD1R_MB - Media Block Read MSD		
Source:	DataPort 1	
Length Bias:	1	
Family:	Other	
Group:	Media Block R/W	
DWord	**Bit**	**Description**
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 04h Format: Opcode Media Block Read message
	13:11	**Reserved** Format: MBZ Ignored
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Media Block Write MSD

MSD1W_MB - Media Block Write MSD		
Source:	DataPort 1	
Length Bias:	1	
Family:	Other	
Group:	Media Block R/W	
DWord	**Bit**	**Description**
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 0Ah Format: Opcode Media Block Write message
	13:11	**Reserved** Format: MBZ Ignored
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Media Transpose Read MSD

MSD1R_TT - Media Transpose Read MSD		
Source:	DataPort 1	
Length Bias:	1	
Family:	Other	
Group:	Transpose Read	
DWord	**Bit**	**Description**
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 00h Format: Opcode Transpose Read message
	13:8	**Reserved** Format: MBZ Ignored
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Memory Fence MSD

MSD_MEMFENCE - Memory Fence MSD

Source: DataPort 0
Length Bias: 1
Family: Other
Group: Memory Fence

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 07h Format: Opcode Memory Fence message
	13	**Commit** Format: Enable Specifies whether control is returned to the thread only after the fence has been honored. Value / Name / Description: 1 / Enabled **[Default]** / The commit writeback register is always required to guarantee ordering. 0 / Reserved / The commit writeback register is always required to guarantee ordering.
	12:9	**L3 Flush** The L3 Flush control is one of the following GSYNC signals. Value / Name / Description: 0h / Disabled **[Default]** / The L3 caches are not flushed. 08h / RW Data / Causes the L3 to flush any RW data. 04h / Constant Data / Causes the L3 to invalidate any Constant data. 02h / Texture Data / Causes the L3 to invalidate any Texture data. 01h / Instructions / Causes the L3 to invalidate all GPU instruction caches.

MSD_MEMFENCE - Memory Fence MSD

		Programming Notes If multiple caches need to be flushed, the commands need to be sent separately. When the memory fence completes, the GSYNC has been started, but may not yet be completed. To know when the GSYNC is completed, Issue any read to the L3 cache after an L3 Flush operation and wait for that data to be returned.
	8	**Reserved** Format: MBZ Ignored
	7:0	**Reserved** Format: MBZ Ignored

MFC_AVC_PAK_OBJECT

MFC_AVC_PAK_OBJECT

Source: VideoCS

Length Bias: 2

The MFC_AVC_PAK_OBJECT command is the second primitive command for the AVC Encoding Pipeline. The same command is used for both CABAC and CAVLC modes. The MV Data portion of the bitstream is loaded as indirect data object.Before issuing a MFC_AVC_PAK_OBJECT command, all AVC MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command.MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice. MFC_AVC_PAK_OBJECT command follows the MbType definition like MFD. Many fields in this command are identical to that in VME output. This is intended to reduce software converting overhead from VME to PAK. Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Length -2 Value / Name: 0009h DWORD_COUNT_n **[Default]** 000Ah DWORD_COUNT_n **[Default]**

MFC_AVC_PAK_OBJECT

1	31:10	**Reserved** Format: MBZ
	9:0	**Indirect PAK-MV Data Length** This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect PAK-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size. Value: [0,512MB) — Name:
3..10	31:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as inline data of the MFC_AVC_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section.
12..23	31:0	**Reserved**

MFC_JPEG_ HUFF_TABLE_STATE

MFC_JPEG_ HUFF_TABLE_STATE

Source: VideoCS

Length Bias: 2

This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image.
Two Huffman tables for luma and chroma will be sent to H/W, and chroma table is used for both U and V.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_JPEG_ HUFF_TABLE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Common Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0AEh Excludes DWord (0,1) Format: =n Total Length - 2
1	31:1	**Reserved** Format: MBZ
	0	**Huff Table ID** Format: U1 Huffman table destination identifier will specify one of two destinations at the encoder into which the Huffman table must be stored. Value / Name / Description

MFC_JPEG_ HUFF_TABLE_STATE

		0 | Huffman table 0 1 | Huffman table 1
2..13	31:0	**DC_TABLE** Format: 3Bytes: Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy 12 categories with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.
14..175	31:0	**AC_TABLE** Format: 3Bytes: Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy 162 run/size with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.

MFC_JPEG_SCAN_OBJECT

MFC_JPEG_SCAN_OBJECT		
Source: VideoCS Length Bias: 2		
Encoder Only		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_JPEG_SCAN_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 001h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:26	**Reserved** Format: MBZ
	25:0	**MCU Count** Format: U26 This field indicates the number of MCUs in the Scan. MCU Count = M_x *X* M_y The number of MCUs in a row: M_x= *(X + (H_1 *8 -1)) / (H_1 *8)* The number of MCUs in a column: M_y= *(Y + (V_1*8 -1)) / (V_1*8)* *X*: The number of samples per line in Y-image *Y*: The number of lines in Y-image *H1*: Horizontal sampling factor of Y-image in the Frame header

MFC_JPEG_SCAN_OBJECT

V1: Vertical sampling factor of Y-image in the Frame header

2 — 31:25 Reserved

Format:	MBZ

24:22 Huffman AC Table

AC Huffman table destination selector specifies one of two possible AC table destinations for each Y, U, V, or R, G, B.
The AC Huffman tables must have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

If AC table 0 is used for Y and AC table 1 is used for U and V, it will be set to 110b. If AC table 0 is used for R, G, and B, it will be set to 000b and so on.
Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit24 (V0)	The third image component must use the AC table 0.
1XXb	Bit24 (V1)	The third image component must use the AC table 1.
X0Xb	Bit23 (U0)	The second image component must use the AC table 0.
X1Xb	Bit23 (U1)	The second image component must use the AC table 1.
XX0b	Bit22 (Y0)	The first image component must use the AC table 0.
XX1b	Bit22 (Y1)	The first image component must use the AC table 1.

Restriction
Restriction : When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR,\ or YUV, **Bit22** is used for the first image component, **Bit23** is used for the second image component, and **Bit24** is used for the third image component.

21 Reserved

Format:	MBZ

20:18 Huffman DC Table

DC Huffman table destination selector specifies one of two possible DC table destinations for each Y, U, V, or R, G, B.
The DC Huffman tables shall have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

if DC table 0 is used for Y and DC table 1 is used for U and V, it will be set to 110b. If DC table 0 is used for R, G, and B, it will be set to 000b and so on.
Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit20 (V0)	The third image component must use the DC table 0.

MFC_JPEG_SCAN_OBJECT

1XXb	Bit20 (V1)	The third image component must use the DC table 1.
X0Xb	Bit19 (U0)	The second image component must use the DC table 0.
X1Xb	Bit19 (U1)	The second image component must use the DC table 1.
XX0b	Bit18 (Y0)	The first image component must use the DC table 0.
XX1b	Bit18 (Y1)	The first image component must use the DC table 1.

Restriction
Restriction : When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR, YUV, **Bit18** is used for the first image component, **Bit19** is used for the second image component, and **Bit20** is used for the third image component.

17 **Head Present Flag**

If this flag is set to 0, then no MFC_JPEG_PAK_INSERT_OBJECT commands will be sent.
If this flag is set to 1, then one or more MFC_JPEG_PAK_INSERT_OBJECT commands will be sent after MFC_JPEG_SCAN_OBJECT command.

Value	Name	Description
0		No insertion into the output bitstream buffer before Scan encoded bitstream
1		Headers, tables, App data insertion into the output bitstream buffer. HW will insert the insertion data before the Scan encoded bitstream.

16 **Is Last Scan**

If this flag is set, then HW will insert EOI (0xFFD9) to the end of Scan encoded bitstream.

Value	Name	Description
0		Not the last Scan.
1		Indicates that the current Scan is the last one.

15:0 **Restart Interval**

Format:	U16

Specifies the number of MCUs in an ECS, except for the last ECS. Restart maker is inserted periodically and it separates the two neighboring ECSs.

Value	Name
0-FFFFh	

Programming Notes
A value of '0' implies that the Scan Data has a single ECS.

MFC_MPEG2_PAK_OBJECT

MFC_MPEG2_PAK_OBJECT

Source: VideoCS

Length Bias: 2

The MFC_MPEG2_PAK_OBJECT command is the second primitive command for the MPEG-2 Encoding Pipeline. Different from AVC, the MV Data portion of the bitstream is loaded as part of MB control data.

Before issuing a MFC_MPEG2_PAK_OBJECT command, all MPEG2_MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command.

MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice.

MFC_ MPEG2_PAK_OBJECT command follows the MbType definition like MFD. Many fields in this command are identical to that in VME output. This is intended to reduce software converting overhead from VME to PAK.

Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_INSERT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h ENC Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0007h Excludes DWord (0,1) Format: =n Total Length - 2

MFC_MPEG2_PAK_OBJECT		
1..8	31:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as inline data of the MFC_MPEG2_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section

MFC_MPEG2_SLICEGROUP_STATE

MFC_MPEG2_SLICEGROUP_STATE

Source: VideoCS

Length Bias: 2

This is a slice group level command and can be issued multiple times within a picture that is comprised of multiple slice groups. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_SLICEGROUP_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h MEDIA_ Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**MbRateCtrlFlag- RateControlCounterEnable (Encoder-only)** To enable the accumulation of bit allocation for rate controlThis field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields.Note: To reset MB level rate control (QRC), we need to set both bits MbRateCtrlFlag and MbRateCtrlReset to 1 in the new slice Value: 0h — Name: Disable Value: 1h — Name: Enable

MFC_MPEG2_SLICEGROUP_STATE

30 — MbRateCtrlReset- ResetRateControlCounter (Encoder-only)

To reset the bit allocation accumulation counter to 0 to restart the rate control.

Value	Name	Description
0h	Disable	Not reset
1h	Enable	reset

29:28 — MbRateCtrlMode- RC Triggle Mode (Encoder-only)

Value	Name	Description
00b		Always Rate Control, whereas RC becomes active if sum_act > sum_target or sum_act < sum_target
01b		Gentle Rate Control, whereas RC becomes active if sum_act > upper_midpt or sum_act < lower_midpt
10b		Loose Rate Control, whereas RC becomes active if sum_act > sum_max or sum_act < sum_min
11b		Reserved

27:24 — MbRateCtrlParam- RC Stable Tolerance (Encoder-only)

Format:	U4

This field specifies the tolerance required to deactivate RC once it has been triggered.

Value	Name
[0, 15]	

23 — RateCtrlPanicFlag - RC Panic Enable (Encoder-only)

If this field is set to 1, RC enters panic mode when sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.

Value	Name
0	Disable
1	Enable

22 — RateCtrlPanicType - RC Panic Type (Encoder-only)

This field selects between two RC Panic methods. If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

Value	Name	Description
0h		QP Panic
1h		CBP Panic

21 — Reserved

Format:	MBZ

MFC_MPEG2_SLICEGROUP_STATE

20	**SkipConvDisabled - MB Type Skip Conversion Disable (Encoder-only)** This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Section 2.3.3.1.6

Value	Name	Description
0h	Enable	Enable skip type conversion
1h	Disable	Disable skip type conversion

19	**IsLastSliceGrp** IsLastSliceGrp = 1 if the current slice group is the last slice group of a picture; 0 otherwise.It is used by the zero filling in the Minimum Frame Size test.
18	**BitstreamOutputFlag - Compressed BitStream Output Disable Flag (Encoder-only)**

Value	Name	Description
0h	Enable	enable the writing of the output compressed bitstream
1h	Disable	disable the writing of the output compressed bitstream

17	**HeaderPresentFlag - Header Insertion Present in Bitstream (Encoder-only)**

Value	Name	Description
0h	Disable	no header insertion into the output bitstream buffer, in front of the current slice encoded bits
1h	Enable	header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.

16	**SliceData PresentFlag - SliceData Insertion Present in Bitstream (Encoder-only)**

Value	Name	Description
0h	Disable	no Slice Data insertion into the output bitstream buffer
1h	Enable	Slice Data insertion into the output bitstream buffer is present.

15	**TailPresentFlag - Tail Insertion Present in bitstream (Encoder-only)**

Value	Name	Description
0h		no tail insertion into the output bitstream buffer, after the current slice encoded bits
1h		tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.

14	**FirstSliceHdrDisabled** when this is on, the first slice header of the slice group is expected to be provided by the user via insertion command. PAK HW will skip it.
13	**IntraSlice** intra slice value included in slice headers, when IntraSliceFlag = 1.
12	**IntraSliceFlag** intra slice flag included in slice headers
11:8	**Reserved**

Format:	MBZ for SliceID extension

MFC_MPEG2_SLICEGROUP_STATE

	7:4	**SliceID[3:0] (Encoder-only)** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP
	3:2	**Reserved** Format: MBZ for StreamID extension
	1:0	**StreamID[1:0] (Encoder-only)** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP
2	31:24	**NextSgMbYcnt - also NextStartVertPos** Vertical count of the first MB in the next slice group (Encoder-only)Note: This field restricts total number of MB in the Y direction to 255 or less.
	23:16	**NextSgMbXcnt - also NextStartHorzPos** BitFieldDesc
	15:8	**FirstMbYcnt - also CurrStartVertPos** Format: U8 also CurrStartVertPos, Vertical count of the first MB in the current slice group (Encoder-only)
	7:0	**FirstMbXcnt - also CurrStartHorzPos** Format: U8 Horizontal count of the first MB in the current slice group (Encoder-only)
3	31:9	**Reserved** Format: MBZ
	8	**SliceGroupSkip** Exists If: //Encoder Only Format: U1 All macroblocks are skipped
	7:6	**Reserved** Format: MBZ
	5:0	**SliceGroupQp** Exists If: //Encoder Only Format: U6 Initial slice quality parameter
4	31:29	**Reserved** Format: MBZ

MFC_MPEG2_SLICEGROUP_STATE

	28:0	**BitstreamOffset - Indirect PAK-BSE Data Start Address (Write)** Exists If: //Encoder Only This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address.It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes.For Write, there is no need to have a data length field. It is assumed the global memory bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access.This field is only valid for AVC encode mode. Value: [0,512MB) — Name:
5	31:24	**MaxQpNegModifier - Magnitude of QP Max Negative Modifier (Encoder-only)** Format: U8 This field specifies the lower limit of the QP modifier. Value: [0, 51] — Name:
	23:16	**MaxQpPosModifier - Magnitude of QP Max Positive Modifier (Encoder-only)** Format: U8 This field specifies the upper limit of the QP modifier. Value: [0, 51] — Name:
	15:12	**ShrinkParam - Shrink Resistance (Encoder-only)** Format: U4 This field specifies the additional points added each time decreased correction is invoked. Value: [0, 15] — Name:
	11:8	**Shrinkaram - Shrink Init (Encoder-only)** Format: U4 This field specifies the initial points required to trip decreased control. Value: [0, 15] — Name:
	7:4	**GrowParam - Grow Resistance (Encoder-only)** Format: U4 This field specifies the additional points added each time increased correction is invoked. Value: [0, 15] — Name:

MFC_MPEG2_SLICEGROUP_STATE

	3:0	**GrowParam - Grow Init (Encoder-only)** Format: U4 This field specifies the initial points required to trip increased control. Value: [0, 15] — Name:
6	31:24	**Reserved** Format: MBZ
	23:20	**CorrectPoints - Correct 6 (Encoder-only)** Format: U4 This field specifies the points used in the lowermost RC region when sum_act <= sum_min. Value: [0, 15] — Name:
	19:16	**CorrectPoints - Correct 5 (Encoder-only)** Format: U4 This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt. Value: [0, 15] — Name:
	15:12	**CorrectPoints - Correct 4 (Encoder-only)** Format: U4 This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target. Value: [0, 15] — Name:
	11:8	**CorrectPoints - Correct 3 (Encoder-only)** Format: U4 This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt. Value: [0, 15] — Name:
	7:4	**CorrectPoints - Correct 2 (Encoder-only)** Format: U4 This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max. Value: [0, 15] — Name:

MFC_MPEG2_SLICEGROUP_STATE

	3:0	**CorrectPoints - Correct 1 (Encoder-only)** Format: U4 This field specifies the points used in the topmost RC region when sum_act > sum_max Value: [0, 15] — Name:
7	31:28	**CV7 - Clamp Value 7 (Encoder-only)** Exists If: //Encoder Only
	27:24	**CV6 - Clamp Value 6 (Encoder-only)** Exists If: //Encoder Only Format: U4
	23:20	**CV5 - Clamp Value 5 (Encoder-only)** Exists If: //Encoder Only Format: U4
	19:16	**CV4 - Clamp Value 4 (Encoder-only)** Exists If: //Encoder Only Format: U4
	15:12	**CV3 - Clamp Value 3 (Encoder-only)** Exists If: //Encoder Only Format: U4
	11:8	**CV2 - Clamp Value 2 (Encoder-only)** Exists If: //Encoder Only Format: U4
	7:4	**CV1 - Clamp Value 1 (Encoder-only)** Exists If: //Encoder Only Format: U4
	3:0	**CV0 - Clamp Value 0 (Encoder-only)** If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped). For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV7	CV6	CV5	CV4	CV3	CV3
none	CV7	CV6	CV5	CV4	CV3	CV3	CV2
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV2
CV6	CV5	CV4	CV3	CV3	CV2	CV2	CV1
CV5	CV4	CV3	CV3	CV2	CV2	CV1	CV1

MFC_MPEG2_SLICEGROUP_STATE

		CV4	CV3	CV3	CV2	CV2	CV1	CV1	CV0
		CV3	CV3	CV2	CV2	CV1	CV1	CV0	CV0
		CV3	CV2	CV2	CV1	CV1	CV0	CV0	CV0

For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV6	CV5	CV4	CV3	CV2	CV1
none	CV7	CV6	CV5	CV4	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV2	CV2	CV1
CV6	CV5	CV4	CV4	CV3	CV2	CV1	CV0
CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0

MFD_AVC_BSD_OBJECT

MFD_AVC_BSD_OBJECT

Source: VideoCS

Length Bias: 2

Description

The MFD_AVC_BSD_OBJECT command is the only primitive command for the AVC Decoding Pipeline. The same command is used for both CABAC and CAVLD modes. The Slice Data portion of the bitstream is loaded as indirect data object.Before issuing a MFD_AVC_BSD_OBJECT command, all AVC states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_AVC_BSD_OBJECT command.

Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_AVC_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 4h — Name: Excludes DWord (0,1) = 0004 **[Default]**

MFD_AVC_BSD_OBJECT

1 — 31:0 — Indirect BSD Data Length

Format:	U32

This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored.
This field must have the same alignment as the Indirect Object Data Start Address.
AVC Short Format : It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data + Emulation Prevention Bytes + any filling trailing zeros after the last MB.
Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.

2 — 31:29 — Reserved

Format:	MBZ

2 — 28:0 — Indirect BSD Data Start Address

Format:	U29

This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the **MFD Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes. In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0. It includes the NAL Header (the NAL Header does not need to perform EMU detection). For AVC Base Layer, it is a single byte. But for MVC, the NAL Header is 4 Bytes long. These NAL Header Unit must be passed to HW in the compressed bitstream buffer.

Value	Name
[0,512MB)	

3..5 — 31:0 — Inline Data

All the required Slice Header parameters and error handling settings are captured as InLine Data of the AVC_BSD_OBJECT command. It has a fixed size of 4 DWs. Its definition is described in the follwoing section: Inline Data Description.

6 — 31:13 — Reserved

Format:	MBZ

6 — 12:9 — Reserved

6 — 8 — AVC Nal Type First Byte Override Bit

Format:	U1

This bit indicates hardware should use the NAL Type (provided below) programmed by driver instead of using the one from bitstream. The NAL byte from bitstream will not be correct.

Value	Name	Description
0	Use Bitstream Decoded NAL Type	NAL Type should come from the first byte of Decoded Bitstream
1	Use Driver Programmed NAL Type	NAL Type should come from "Driver Provided Nal Type" programmed by driver.

MFD_AVC_BSD_OBJECT		
	7:0	**Driver Provided Nal Type Value** Format: U8 This will replace the first byte of the NAL unit, containing forbidden_zero_bit, nal_ref_idc, and nal_unit_type. **Programming Notes** This byte should be ignored if "AVC NAL Type First Byte Override Bit" is programmed to 0

MFD_AVC_DPB_STATE

MFD_AVC_DPB_STATE

Source: VideoCS

Length Bias: 2

This is a frame level state command used only in DXVA2 AVC Short Slice Bitstream Format VLD mode.
RefFrameList[16] of DXVA2 interface is replaced with intel Reference Picture Addresses[16] of MFX_PIPE_BUF_ADDR_STATE command. The LongTerm Picture flag indicator of all reference pictures are collected into LongTermPic_Flag[16].
FieldOrderCntList[16][2] and CurrFieldOrderCnt[2] of DXVA2 interface are replaced with intel POCList[34] of MFX_AVC_DIRECTMODE_STATE command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]**

MFD_AVC_DPB_STATE

1	31:16	**LongTermFrame_Flag[16][1 bit]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame. **Value** / **Name** 1 – the picture is a long term reference picture 0 – the picture is a short term reference picture
	15:0	**Non-ExistingFrame_Flag[16][1 bit]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame. **Value** / **Name** / **Description** 1 / INVALID / the reference picture in that entry of RefFrameList[] does not exist anymore. 0 / VALID / the reference picture in that entry of RefFrameList[] is a valid reference **Programming Notes** When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the corresponding bit of NonExistingFrameFlags shall be set to 0.
2	31:0	**UsedForReference_Flag[16][2 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 2 bits per reference frame. **Value** / **Name** / **Description** 0 / NOT_REFERENCE / indicates a frame is "not used for reference". 1 / TOP_FIELD / bit[0] indicates that the top field of a frame is marked as "used for reference". 2 / BOTTOM_FIELD / bit[1] indicates that the bottom field of a frame is marked as "used for reference". 3 / FRAME / bit[1:0] indicates that a frame (or field pair) is marked as "used for reference".
3..10	31:0	**LTSTFrameNumList[16][16 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame.Depending on the corresponding LongTermFrame_Flag[], the content of this field is interpreted differently. **Value** / **Name** / **Description** 1 / LongTermFrame_Flag[i] / LTSTFrameNumList[i] represent LongTermFrameIdx. 0 / ShortTermFrame_Flag[i] / LTSTFrameNumList[i]represent Short Term Picture FrameNum. **Programming Notes** When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.

MFD_AVC_DPB_STATE		
11..18	31:0	**ViewIDList[16][16 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. The view ids are 10-bits, the upper 6 bits are ignored."000000" & ViewId1[9:0] & "000000" & ViewId0[9:0] **Programming Notes** When an Intel RefFrameList[i] is not an valid entries, Viewid should be set to 0x00
19..22	31:0	**ViewOrderListL0[16][8 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored. 0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL0[i] is not an valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.
23..26	31:0	**ViewOrderListL1[16][8 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored. 0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL1[i] is not an valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.

MFD_AVC_PICID_STATE

MFD_AVC_PICID_STATE

Source: VideoCS

Length Bias: 2

This is a frame level state command used for both AVC Long and Short Format in VLD mode.PictureID[16] contains the pictureID of each reference picture (16 maximum) so hardware can uniquely identify the reference picture across frames (this will be used for DMV operation).This command will be needed for both short and long format.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MFD_AVC_DPB_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h DEC Format: OpCode
	20:16	**SubOpcode B** Default Value: 5h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0008h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:1	**Reserved** Format: MBZ

<table>
<tr><th colspan="3">MFD_AVC_PICID_STATE</th></tr>
<tr><td></td><td>0</td><td>PictureID Remapping Disable
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>AVC decoder will use 16 bits Picture ID to handle DMV and identify the reference picture</td><td>Desc</td></tr>
<tr><td>1h</td><td>AVC decoder will use 4 bits FrameStoreID (index to RefFrameList) to handle DMV and identify the reference picture.</td><td>Desc</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If Picture ID Remapping Disable is "1", PictureIDList will not be used.</td></tr>
</table></td></tr>
<tr><td>2..9</td><td>31:0</td><td>PictureIDList[16][16 bits]
One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. PictureID of each Frame uniquely identifies the reference picture across frames. The same number cannot be reused until the referece picture is completely retired(no longer used for referece)When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.</td></tr>
</table>

MFD_AVC_SLICEADDR

MFD_AVC_SLICEADDR

Source: VideoCS

Length Bias: 2

This is a Slice level command used only for DXVA2 AVC Short Slice Bitstream Format VLD mode.When decoding a slice, H/W needs to know the last MB of the slice has reached in order to start decoding the next slice. It also needs to know if a slice is terminated but the last MB has not reached, error conealment should be invoked to generate those missing MBs. For AVC DXVA2 Short Format, the only way to know the last MB position of the current slice, H/W needs to snoop into the next slice's start MB address (a linear address encoded in the Slice Header). Since each BSD Object command can have only one indirect bitstream buffer address, this command is added to help H/W to snoop into the next slice's slice header and retrieve its Start MB Address. This command will take the next slice's bitstream buffer address as input (exactly the same way as a BSD Object command), and parse only the first_mb_in_slice syntax element. The result will stored inside the H/W, and will be used to decode the current slice specified in the BSD Object command.Only the very first few bytes (max 5 bytes for a max 4K picture) of the Slice Header will be decoded, the rest of the bitstream are don't care. This is because the first_mb_in_slice is encoded in Exponential Golomb, and will take 33 bits to represent the max 256 x 256 = 64K-1 value. The indirect data of MFD_AVC_SLICEADDR is a valid BSD object and is decoded as in BSD OBJECT command.The next Slice Start MB Address is also exposed to the MMIO interface.The Slice Start MB Address (first_mb_in_slice) is a linear MB address count; but it is translated into the corresponding 2D MB X and Y raster position, and are stored internally as NextSliceMbY and NextSliceMbX.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_AVC_ SLICEADDR Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 7h Format: OpCode
	15:12	**Reserved** Format: MBZ

MFD_AVC_SLICEADDR

	11:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** 1h: Excludes DWord (0,1) **[Default]**
1	31:0	**Indirect BSD Data Length** Format: U32 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. Driver always programs this up to 5 bytes; for bitstream less than 5 bytes, driver program the lesser value. (Emulation Prevention Byte should never happen for the first 5 bytes when the max picture size can only be 4Kx4K)It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect BSD Data Start Address** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes.In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0.It includes the NAL Header Byte. (but does not perform EMU detection).Must provide a valid MB address, even if error. MB must be clamped to within a pic boundary. **Value** / **Name** [0,512MB)
3	31:13	**Reserved** Format: MBZ
	12:9	**Reserved**
	8	**AVC NAL Type First Byte Override Bit** Format: U1 This bit indicates hardware should use the NAL Type (provided below) programmed by driver instead of using the one from bitstream. The NAL byte from bitstream will not be correct. **Value** / **Name** / **Description** 0 / Use Bitstream Decoded NAL Type / NAL Type should come from first byte of decoded bitstream. 1 / Use Driver Programmed NAL Type / NAL Type should come from "Driver Provided NAL Type Values" programmed by driver.

MFD_AVC_SLICEADDR		
	7:0	**Driver Provided NAL Type Value** Format: U8 This will replace the first byte of the NAL unit, containing forbidden_zero_bit, nal_ref_idc, and nal_unit_type. **Programming Notes** This byte should be ignored if "AVC NAL Type First Byte Override Bit" is programmed to 0

MFD_IT_OBJECT

MFD_IT_OBJECT

Source: VideoCS

Length Bias: 2

All weight mode (default and implicit) are mapped to explicit mode. But the weights come in either as explicit or implicit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_IT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 06h Excludes DWord (0,1) For AVC = Ch Format: =n Total Length - 2 Note: Regardless of the mode, inline data must be present in this command.
1	31:10	**Reserved** Format: MBZ

MFD_IT_OBJECT

	9:0	**Indirect IT-MV Data Length** Format: U10 FormatDesc: In bytes This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.AVC-IT Mode: It must be DWord aligned (since each MV is 4bytes in size)Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data).
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect IT-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into the IT pipeline for processing. This pointer is relative to the Indirect IT-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect MV Data Length is set to 0. Alignment of this address depends on the mode of operation.AVC-IT Mode: It must be DWord aligned (since each MV is 4 bytes in size). This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data). **Value** / **Name** [0,512MB)
3	31:12	**Reserved** Format: MBZ
	11:0	**Indirect IT-COEFF Data Length** This field provides the length in bytes of the indirect data, which contains all the non-zero coefficients for the current MB. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-COEFF Data Start Address field is ignored. Since each IT-COEFF data is 1 DW in size, with 12 bits, this field can be extended to support up to 4:4:4 format.(256 pixel * 3 byte pixel components * 4 bytes per coeff).This field must be integer multiple of 16-bytes for AVC (since each coefficient is 4 bytes in size).This field is only valid in AVC, VC1, MPEG2 decoder IT mode. **Value** / **Name** [0,3072] / In bytes [0, 256*3*4]
4	31:29	**Reserved** Format: MBZ

MFD_IT_OBJECT

	28:0	**Indirect IT-COEFF Data Start Address Offset** This field specifies the memory starting address (offset) of the coeff data to be loaded into the IT pipeline for processing. This pointer is relative to the Indirect IT-COEFF Object Base Address.Hardware ignores this field if indirect IT-COEFF data is not present, i.e. the Indirect IT-COEFF Data Length is set to 0.This field must be DW aligned, since each coeff icient is 4 bytes in size.Driver will determine the Num of EOB 4x4/8x8 must match the block cbp flags, if not match, hardware cannot hang - add error handling.This field is only valid in AVC, VC1, MPEG2 decoder IT mode. Value: [0,512MB) — Name:
5	31:6	**Reserved** Format: MBZ
	5:0	**Indirect IT-DBLK Control Data Length** Format: U6 This field provides the length in bytes of the indirect data, which contains all the deblocker control information for the current MB (in 4x4 sub-block partitioning). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-DBLK Data Start Address field is ignored. This field must have the same alignment as the Indirect IT-DBLK Data Start Address. It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size.This field is only valid in AVC decoder IT mode.
6	31:29	**Reserved** Format: MBZ
	28:0	**Indirect IT-DBLK Control Data Start Address Offset** Format: IndirectObjectBaseAddress[28:0] This field specifies the memory starting address (offset) of the Deblocker control data to be fetched into the IT Pipeline for processing. This pointer is relative to the Indirect IT-DBLK Object Base Address. Hardware ignores this field if indirect data is not present, ie. The indirect IT-DBLK Control Data Length is set to 0. It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size. This field is only valid in AVC decoder IT mode. Value: [0,512MB) — Name:
7..n	31:0	**Inline Data** Union for all 3 codecs Includes IT, MC, IntraPred inline data as well as Deblocker control information AVC-IT Modes: Hardware interprets this data in the specified format. VC1-IT Modes: Hardware interprets this data in the specified format. MV inline MPEG2-IT Modes: Hardware interprets this data in the specified format. (IS mode) MV inline For AVC there 7 DWords of inline data, hence N is equal to 13.

MFD_JPEG_BSD_OBJECT

MFD_JPEG_BSD_OBJECT

Source: VideoCS
Length Bias: 2
Exists If: //Decoder

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_JPEG_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 004h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Indirect Data Length** . It is the length in bytes of the bitstream data for the current Scan. It includes the first byte of the first MCU and the last non-zero byte of the last MCU in the Scan. Specifically, the zero-padding bytes (if present) are excluded. Hardware ignores the contents after the last non-zero byte.
2	31:29	**Reserved** Format: MBZ

MFD_JPEG_BSD_OBJECT

	28:0	**Indirect Data Start Address** Format: IndirectObjectOffset[28:0] This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the JPEG bitstream data
3	31:29	**Reserved** Format: MBZ
	28:16	**Scan Horizontal Position** Format: U13 bits in blocks This field indicates the horizontal position (in block units) of the first MCU in the Scan.
	15:13	**Reserved** Format: MBZ
	12:0	**Scan Vertical Position** Format: U13 bits in blocks This field indicates the vertical position (in block units) of the first MCU in the Scan.
4	31	**Reserved** Format: MBZ
	30	**Interleaved** Value / Name / Description 0 / Non-Interleaved / one component in the Scan 1 / Interleaved / multiple components in the Scan
	29:27	**Scan Components** Bit0: Y Bit1: U Bit2: V For example, if non-interleaved Y, then it will be set to 001b. If interleaved Y, U, and V, it will be set to 111b.
	26	**Reserved** Format: MBZ
	25:0	**MCU Count** Format: U26 This field indicates the number of MCUs in the Scan.
5	31:16	**Reserved** Format: MBZ

MFD_JPEG_BSD_OBJECT		
	15:0	**RestartInterval(16 bit)** Format: U16 Specifies the number of MCU in restart interval. Valid values are 1->0xFFFFValue of 0 implies that all the SCAN have only one ECS.

MFD_MPEG2_BSD_OBJECT

MFD_MPEG2_BSD_OBJECT

Source: VideoCS

Length Bias: 2

Different from AVC and VC1, MFD_MPEG2_BSD_OBJECT command is pipelinable. This is for performance purpose as in MPEG2 a slice is defined as a group of MBs of any size that must be within a macroblock row.Slice header parameters are passed in as inline data and the bitstream data for the slice is passed in as indirect data. Of the inline data, slice_horizontal_position and slice_vertical_position determines the location within the destination picture of the first macroblock in the slice. The content in this command is identical to that in the MEDIA_OBJECT command in VLD mode described in the Media Chapter.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_MPEG2_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n Total Length - 2

MFD_MPEG2_BSD_OBJECT

1	31:0	**Indirect BSD Data Length** Format: U32 It is the length in bytes of the bitstream data for the current slice. It includes the first byte of the first macroblock and the last non-zero byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. This field is sized to support beyond MPEG-2 MP@HL bitstream (<4K). According to Table 8-6 of ISO/IEC 13818-2, the maximum number of bits per macroblock for 4:2:0 is 4608. So the maximum slice size for 4K x 4K is 4608 * 256 / 8 = 147,456 bytes (0x24000), which requires 18 bits. **Programming Notes** As MPEG-2 spec does not post any limitation of the size of zero-padding bytes, it is possible to have a slice data with large length (including zero-padding bytes). As the data beyond 0x10E00 would only be zero bytes for a valid slice data zero-padding restriction is removed
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect Data Start Address** Format: IndirectObjectOffset[28:0] This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the MPEG2 VLD bitstream data This address points to the first byte of the MB layer data, i.e. not including slice header.
3..4	31:0	**Inline Data** All the required Slice Header parameters and error handling settings are captured as MPEG2_BSD_OBJECT Inline Data Descriptor structures. It has a fixed size of 2 DWs. Its definition is described in the next section.

MFD_VC1_BSD_OBJECT

MFD_VC1_BSD_OBJECT

Source: VideoCS

Length Bias: 2

The MFD_VC1_BSD_OBJECT command is the only primitive command for the VC1 Decoding Pipeline. The macroblock data portion of the bitstream is loaded as indirect data object.Before issuing a MFD_VC1_BSD_OBJECT command, all VC1 states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VC1_BSD_OBJECT command.VC1 deblock filter kernel cross the slice boundary if in the last MB row of a slice, so need to know the last MB row of a slice to disable the edge mask. There is why VC1 BSD hardware need to know the end of MB address for the current slice. As such no more phantom slice is needed for VC1, as long as the driver will program both start MB address in the current slice and the start MB address of the next slice. As a result, we can also support multiple picture state commands in between slices.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:24	**Reserved** Format: MBZ

MFD_VC1_BSD_OBJECT

	23:0	**Indirect BSD Data Length** Format: U24 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.GEN6 Long Format : It is the length in bytes of the bitstream data for the current slice/picture. It includes the first byte of the first macroblock and the last byte of the last macroblock in the slice/picture. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte (trailing zeros). This field is sized to support VC1 AP@L4 Level bitstream. It includes the byte that contains the First MB Bit OffsetGEN7 Short Format : It is the length in bytes of the bitstream data for the current slice, including Picture/Slice Header + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect Data Start Address** Format: IndirectObjectOffset[28:0] This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VC1 bitstream data. Value: [0,512MB) — Name:
3	31:24	**Reserved** Format: MBZ
	23:16	**Slice Start Vertical Position** This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. For SecondField this value is reset to zero as oppoed to the VC1 spec Ref: 9.1.2 Slice Layer.This field is for both Long and Short VC1 Interface Format.
	15:9	**Reserved** Format: MBZ
	8:0	**Next Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks.This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering)This field is maintained and provided by the driver for both Long and Short VC1 Interface Format.
4	31:16	**First_MB_Byte_Offset of Slice Data or Slice Header** For DXVA2 VC1 Short Format onlyIt gives the byte offset to locate the first MB data in the bitstream for a slice, relative to the Indirect BSD Data Start Address.

MFD_VC1_BSD_OBJECT

	15:5	**Reserved** Format: MBZ
	4	**Emulation Prevention Byte Present** 0h: H/W needs to perform Emulation Byte Removal 1h: H/W does not need to perform Emulation Byte Removal
	3	**Reserved** Format: MBZ
	2:0	**FirstMbBitOffset (First Macroblock Bit Offset)** Format: U3 This field provides the bit offset of the first macroblock of the Slice in the first byte of the input compressed bitstream.It is used with First_MB_Byte_Offset for non-byte aligned position.

MFD_VC1_LONG_PIC_STATE

MFD_VC1_LONG_PIC_STATE

Source: VideoCS

Length Bias: 2

MFX_VC1_LONG PIC_STATE command encapsulates the decoding parameters that are read or derived from bitstream syntax elements above (inclusive) picture header layer. These parameters are static for a picture and when slice structure is present, these parameters are not changed from slice to slice of the same picture. Hence, this command is only issued at the beginning of processing a new picture and prior to the VC1_*_OBJECT command. The values set for these state variables are retained internally across slices.Only the parameters needed by hardware (BSD unit) to decode bit sequence for the macroblocks in a picture layer or a slice layer are presented in this command. Other parameters such as the ones used for inverse transform or motion compensation are provided in MFX_VC1_PRED_PIPE_STATE command.This Long interface format is intel proprietary interface. Driver will need to perform addition operations to generate all the fields in this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_LONG_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0004h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:24	**Reserved** Format: MBZ

MFD_VC1_LONG_PIC_STATE

	23:16	**PictureHeightInMBsMinus1 (Picture Height Minus 1 in Macroblocks)** Format: U8 This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).This field is used in VLD and IT modes. **Programming Notes** Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boudary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.
	15:8	**Reserved** Format: MBZ
	7:0	**PictureWidthInMBsMinus1 (Picture Width Minus 1 in Macroblocks)** Format: U8-1 This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16).This field is used in VLD and IT modes
2	31:24	**Bitplane Buffer Pitch Minus 1** Format: U7-1 Pitch in (Bytes - 1). Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format (Gen6 and Gen7), it is written by an application and later read by the HW. But in VC1 Short Format (Gen7 only), it is written and read by H/W only.This field is specified for better performance **Value** / **Name** [0, FFFFFFFFh] / **Programming Notes** For Gen6 : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Long Format : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row.This field is not used in IT mode, used in VLD mode only.For VC1 DXVA2 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.
	23:16	**Reserved** Format: MBZ

MFD_VC1_LONG_PIC_STATE

	15	**DmvSurfaceValid** Indicated when the DMV read surface is valid. This surface stored the direct motion vectors and Mb type.This field is set for B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture).Whne the current picture being decoded is an I, P or BI, this bit is set to 0, since there is no DMV read in these picture decoding process.This field is not used in IT mode, used in VLD mode only.
	14	**ImplicitQuantizer** Derived by driver from QUANTIZER.This field is used in intel VC1 VLD Long Format only, not used in IT and DXVA2 VC1.This bit is set to 1 when syntax element QUANTIZER=0, else its set to 0
	13	**Interpolation Rounder Contro** Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process.This field is used in VLD and IT modes. **Programming Notes** This bit field is taken from bRcontrol in DXVA_PictureParameters data structure
	12	**SyncMarker** Indicates whether sync markers are enabled/disabled. If enable, sync markers "may be" present in the current video sequence being decoded. It is a sequence level syntax element and is valid only for Simple and Main Profiles. **Value** / **Name** / **Description** 0h / Not Present / Sync Marker is not present in the bitstream 1h / Maybe present / Sync Marker maybe present in the bitstream **Programming Notes** This field is only valid in VLD mode.For Simple Profile, SyncMarker must set to 0.For Main Profile, SyncMarker can be set to 0 or 1.This field is used in both intel and MS DXVA2 VLD interface, but not used in IT mode.
	11:8	**Motion Vector Mode** This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision.Before the polarity of Chroma Half-pel or Q-pel is reversed from DXVA2 Spec, now I have fixed it to match with DXVA2 VC1 Spec. **Value** / **Name** / **Description** 0XX0b / / Chroma Quarter -pel + Luma bicubic. (can only be 1MV) 0XX1b / / Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV) 1XX0b / / Chroma Quarter -pel + Luma bilinear. (can only be 1MV) 1XX1b / / Chroma Half-pel + Luma bilinear

MFD_VC1_LONG_PIC_STATE

Programming Notes
Bits 11:8 are taken from bMVprecisionAndChromaRelation in DXVA_PictureParameters data structure.Bit 11 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 8 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion.This field is used in both VLD and IT modes.

7 **RangeReductionScale**

This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled.

Value	Name	Description
0h		Scale down reference picture by factor of 2
1h		Scale up reference picture by factor of 2

Programming Notes
This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

6 **RangeReduction Enable**

This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element (DXVA_PictureParameters bPicDeblocked bit 5) in the Picture Header.

Value	Name	Description
0h	Disable	Range reduction is not performed
1h	Enable	Range reduction is performed

Programming Notes
This field is for Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture.RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

MFD_VC1_LONG_PIC_STATE

	5	**LOOPFILTER Enable Flag** This filed is the decoded syntax element LOOPFILTER in bitstream. It indicates if In-loop Deblocking is ON according to picture level bitstream syntax control. This bit affects BSD unit and also the loop filter unit.When this bit is set to 1, PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command must also be set to 1. In this case, in-loop deblocking operation follows the VC1 standard - deblocking doesn't cross slice boundary.When this bit is set to 0, but PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command is set to 1. It indicates the loop filter unit is used for out-of-loop deblocking. In this case, deblocking operation does cross slice boundary.This field is used in VLD mode only, not in IT mode.

Value	Name	Description
0h	Disable	Disables loop filter
1h	Enable	Enables loop filter

	4	**Overlap Smoothing Enable Flag** This field is the decoded syntax element OVERLAP in bitstreamIndicates if Overlap smoothing is ON at the picture levelThis field is used in both VLD and IT modes.

Value	Name	Description
0h	Disable	to disable overlap smoothing filter
1h	Enable	to enable overlap smoothing filter

	3	**Secondfield** This flag is set for the second field in field pictures.This field is used in both VLD and IT modes.
	2:1	**Reserved**

Format:	MBZ

	0	**VC1 Profile** specifies the bitstream profile.This field is used in both VLD and IT modes.

Value	Name	Description
0h	Disable	current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile)
1h	Enable	current picture is in Advanced Profile

Programming Notes
This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not.

3	31	**Reserved**

Format:	MBZ

	30:29	**CondOver** This field is the decoded syntax element CONDOVER in a bitstream of advanced profile. It controls the overlap smoothing filter operation for an I frame or an BI frame when the picture level qualization step size PQUANT is 8 or lower.This field is used in intel VC1 VLD mode only, not in DXVA2 VC1 and IT modes.

Value	Name	Description

MFD_VC1_LONG_PIC_STATE

00b		No overlap smoothing
01b		Reserved
10b		Always perform overlap smoothing filter
11b		Overlap smoothing on a per macroblock basis based on OVERFLAGS

28:26 **PicType (Picture Type)**

This field specifies the coding type of the picture according to the Frame Coding Mode. When FCM = 00 | 01 (a Progressive or Interlaced Frame Picture):000 = I001 = P010 = B011 = BI100 = SkippedOther encodings are reservedWhen FCM = 10 | 11 (a Field Picture)000 = I/I001 = I/P010 = P/I011 = P/P100 = B/B101 = B/BI110 = BI/B111 = BI/BIAlthough, for a field picture, it is set for a field-pair, but HW will only look at one field state only, and the other field state is don't care. This field is read and qualified with the SecondField flag internally.This field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For DXVA2 VC1 IT mode, driver needs to convert the DXVA2 interface to intel HW VLD Long Format interface.

25:24 **FCM (Frame Coding Mode)**

This is the same as the variable FCM defined in VC1.This field must be set to 0 for Simple and Main ProfilesThis field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For DXVA2 VC1 IT mode, driver needs to convert the DXVA2 interface to intel HW VLD Long Format interface.

Value	Name	Description
00b	Disable	Progressive Frame Picture
01b	Enable	Interlaced Frame Picture
10b		Field Picture with Top Field First
11b		Field Picture with Bottom Field First

23:21 **Reserved**

Format:	MBZ

20:16 **AltPQuant (Alternative Picture Quantization Value)**

This field is identical to the variable ALTPQUANT which is derived from VOPDQUANT configuration in the VC1 standard.This field must be set to 0 for Simple/Main I and BI pictures as VOPDQUANT is not present.This field is used in intel VC1 VLD Long Format mode only, not used in DXVA2 VC1 VLD and IT modes.

15:13 **Reserved**

Format:	MBZ

12:8 **PQuant (Picture Quantization Value)**

Format:	U5

This is the same as the calculated variable PQUANT in VC1 standard where PQuant = PQINDEX, except when QUANTIZER = 0 and PQINDEX > 8, it is given asPQuant = (PQINDEX < 29) ? PQINDEX - 3 : PQINDEX*2 - 31This field is used in all picture types (I, P, B and BI) and all operating modes (IT mode and intel and DXVA2 VLD modes).

MFD_VC1_LONG_PIC_STATE

	7:0	**BScaleFactor** BScaleFactorThis field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1 standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21 encodings of BFRACTION as shown in the table here. Other values are reserved.MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine Motion Prediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "BScaleFactor >= 128".This field is only valid for B pictures. This field is used only in intel VC1 VLD Long format mode, it is not used in DXVA2 VC1 VLD and IT modes.BFRACTION VLCBFRACTIONBScaleFactor0001/21280011/3850102/31700111/4641003/41921011/5511102/510211100003/515311100014/520411100101/64311100115/621511101001/73711101012/77411101103/711111101114/714811110005/718511110016/722211110101/83211110113/89611111005/816011111017/8224
4	31:30	**Reserved** Format: MBZ
	29:28	**UnifiedMvMode (Unified Motion Vector Mode)** This field is a combination of the variables MVMODE and MVMODE2 in the VC1 standard, for parsing Luma MVD from the bitstream. This field is used to signal 1MV vs 4MVallowed (Mixed Mode). This field is also used to signal Q-pel or Half-pel MVD read from the bitstream. The bicubic or bilinear Luma MC interpolation mode is duplicate information from Motion Vector Mode field, and is ignored here.This field is used in intel VC1 VLD Long Format mode only, it is not used in DXVA2 VC1 VLD and IT modes.

Value	Name	Description
00b		Mixed MV, Q-pel bicubic
01b		1-MV, Q-pel bicubic
10b		1-MV half-pel bicubic
11b		1-MV half-pel bilinear

	27	**FourMvSwitch (Four Motion Vector Switch)** This field indicates if 4-MV is present for an interlaced frame P picture. It is identical to the variable 4MVSWITCH (4 Motion Vector Switch) in VC1 standard.This field is used in intel VC1 VLD Long Format mode only, it is not used in DXVA2 VC1 VLD and IT modes.

Value	Name	Description
0h	Disable	only 1-MV
1h	Enable	1, 2, or 4 MVs

	26	**FastUVMCFlag (Fast UV Motion Compensation Flag)** This field specifies whether the motion vectors for UV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard.This field is used in both VLD and IT modes.It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector Mode LSBit.

Value	Name	Description
0h		no rounding

MFD_VC1_LONG_PIC_STATE

1h		quarter-pel offsets to half/full pel positions

25 RefFieldPicPolarity (Reference Field Picture Polarity)

This field specifies the polarity of the one reference field picture used for a field P picture. It is derived from the variable REFFIELD defined in VC1 standard and is only valid when one field is referenced (NUMREF = 0) for a field P picture.When NUMREF = 0 and REFFIELD = 0, this field is the polarity of the reference I/P field that is temporally closest; When NUMREF = 0 and REFFIELD = 1, this field is the polarity of the reference I/P field that is the second most temporally closest. The distance is measured based on display order but ignoring the repeated field if present (due to RFF = 1).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Top (even) field
1h		Bottom (odd) field

24 NumRef (Number of References)

This field indicates how many reference fields are referenced by the current (field) picture. It is identical to the variable NUMREF in the VC1 standard. This field is only valid for field P picture (FCM = 10 | 11).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		One field referenced
1h		Two fields referenced

23:20 BwdRefDist (Reference Distance)

This field is valid only in B field pictures giving the value of BRFD. The field is ignored in P Picture.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

19:16 FwdRefDist (Reference Distance)

Format:	U4

This field is the number of frames between the current frame and its reference frame. It is derived from the syntax element REFDIST (P Reference Distance) in the VC1 standard. 0 means that the previous frame is the reference frame.It has the same value as of FRFD for both P and B field pictures.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name
[0, 15]	

15:12 Reserved

Format:	MBZ

11:10 ExtendedDMVRange (Extended Differential Motion Vector Range Flag)

This field specifies the differential motion vector range in interlaced pictures. It is equivalent to the variable DMVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

MFD_VC1_LONG_PIC_STATE

Value	Name	Description
00b		No extended range
01b		Extended horizontally
10b		Extended vertically
11b		Extended in both directions

9:8 **ExtendedMVRange (Extended Motion Vector Range Flag)**

This field specifies the motion vector range in quarter-pel or half-pel modes. It is equivalent to the variable MVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes

Value	Name	Description
00b		[-256, 255] x [-128, 127]
01b		512, 511] x [-256, 255]
10b		[-2048, 2047] x [-1024, 1023]
11b		[-4096, 4095] x [-2048, 2047]

7:4 **AltPQuantEdgeMask (Alternative Picture Quantization Edge Mask)**

This field is a bit mask for the four edges in clock-wise order, indicating whether AltPQuant is used for the edge macroblocks.It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found..This field is valid only if AltPQuantConfig is 01. Bit 0: Left picture edge macroblocksBit 1: Top picture edge macroblocksBit 2: Right picture edge macroblocksBit 3: Bottom picture edge macroblocksThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

3:2 **AltPQuantConfig (Alternative Picture Quantization Configuration)**

This field specifies the way AltPQuant is used in the picture. It determines how to compute the macroblock quantizer step size, MQUANT. It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found..This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
00b		AltPQuant not used
01b		AltPQuant is used and applied to edge macroblocks only
10b		MQUANT is encoded in macroblock layer
11b		AltPQuant and PQuant are selected on macroblock basis

1 **HalfQP**

This field is used for inverse quantization of AC coefficients. It is valid only when PQuant is used.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

0 **PQuantUniform**

Indicating if uniform quantization applies to the picture. It is used for inverse quantization of the AC coefficients.QUANTIZER 001123PQUANTIZER - -01--PQINDEX>=9<=8----

MFD_VC1_LONG_PIC_STATE

PQuantUniform010201ImplicitQuantizer = 0, and PQuantUniform = 0 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=0; and 2) QUANTIZER = 10b.ImplicitQuantizer = 0, and PQuantUniform = 1 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=1; and 2) QUANTIZER = 11bThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-uniform
1h		Uniform

5 | 31 **BitplanePresentFlag (Bitplane Buffer Present Flag)**

This field indicates whether the bitplane buffer is present for the picture. If set, at least one of the fields listed in bits 22:16 is coded in non-raw mode, and Bitplane Buffer Base Address field in the VC1_BSD_BUF_BASE_STATE command points to the bitplane buffer. Otherwise, all the fields that are applicable for the current picture in bits 22:16 must be coded in raw mode.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		bitplane buffer is not present
1h		bitplane buffer is present

30 **ForwardMbRaw**

This field indicates whether the FORWARDMB field is coded in raw or non-raw mode.This field is only valid when PictureType is B.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		non-raw mode
1h		raw mode

29 **MvTypeMbRaw**

This field indicates whether the MVTYPREMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

28 **SkipMbRaw**

This field indicates whether the SKIPMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P or B.0 = non-raw mode1 = raw modeThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

MFD_VC1_LONG_PIC_STATE

	27	**DirectMbRaw** This field indicates whether the DIRECTMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P or B.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

	26	**OverflagsRaw** This field indicates whether the OVERFLAGS field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

	25	**AcPredRaw** This field indicates whether the ACPRED field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

	24	**FieldTxRaw** This field indicates whether the FIELDTX field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

	23	**Reserved**

Format:	MBZ

	22:20	**MvTab (Motion Vector Table)**

Format:	U3

This field specifies which motion vector table(s) is (are) used for motion vector (differential) decoding in a P or B picture. This field is the combination of the variables MVTAB and IMVTAB in the VC1 standard. Two bits are defined for progressive frame pictures; And two or three bits are defined for interlaced field/frame pictures depending on NUMREF and P or B picture types.This field is valid for P and B pictures. It is not valid for I pictures.For P or B progressive frame pictures0 = Motion Vector Differential VLD Table 01 = Motion Vector Differential VLD Table 12 = Motion Vector Differential VLD Table 23 = Motion Vector Differential VLD Table 3The other encodings are reservedFor P interlace field pictures with NUMREF = 0 or P/B interlace frame

MFD_VC1_LONG_PIC_STATE

		pictures0 = 1-Reference Table 01 = 1-Reference Table 12 = 1-Reference Table 23 = 1-Reference Table 3The other encodings are reservedFor P interlace field picture with NUMREF = 1 or B interlaced field pictures0 = 2-Reference Table 01 = 2-Reference Table 12 = 2-Reference Table 23 = 2-Reference Table 34 = 2-Reference Table 45 = 2-Reference Table 56 = 2-Reference Table 67 = 2-Reference Table 7The other encodings are reservedThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	19:18	**FourMvBpTab (4-MV Block Pattern Table)** This field specifies which table is used to decode the 4-MV block pattern (4MVBP) syntax element in 4-MV macroblocks. It is identical to the variables 4MVBPTAB in the VC1 standard, section 9.1.1.37. This field is valid only in interlace frame P, B pictures, or interlace field P, B pictures. It is not valid for I picture.For interlace field P and B pictures, it is only valid if UnifiedMvMode is equal to Mixed-MV Type. For interlace frame P picture, it is only valid if FourMvSwitch is 1.For interlace frame B picture, it is always valid.0 = 4MVBP Table 01 = 4MVBP Table 12 = 4MVBP Table 23 = 4MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	17:16	**TwoMvBpTab (2MV Block Pattern Table)** This field specifies which table is used to decode the 2MV block pattern (2MVBP) syntax element in 2MV field macroblocks. It is identical to the variables 2MVBPTAB in the VC1 standard, section 9.1.1.36. This field is valid only in interlace frame P/B pictures. It is not valid for I picture, nor for interlace field P or B pictures.0 = 2MVBP Table 01 = 2MVBP Table 12 = 2MVBP Table 23 = 2MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	15:14	**Reserved** Format: MBZ
	13:12	**TransType (Picture-level Transform Type)** Format: U2 This field specifies the Transform Type at picture level. It is identical to the variable TTFRM in the VC1 standard, section 7.1.1.41.This field is only valid when TransTypeMbFlag is 1. Otherwise, it is reserved and MBZ.This field is set to 00 when VSTRANSFORM is 0 in the entry point layer.00 = 8x8 Transform01 = 8x4 Transform10 = 4x8 Transform11 = 4x4 TransformThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	11	**TransTypeMbFlag (Macroblock Transform Type Flag)** This field indicates whether Transform Type is fixed at picture level or variable at macroblock level. It is identical to the variable TTMBF in the VC1 standard, section 7.1.1.40.This field is set to 1 when VSTRANSFORM is 0 in the entry point layer.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		variable transform type in macroblock layer
1h		use picture level transform type TransType

MFD_VC1_LONG_PIC_STATE

	10:8	**MbModeTab (Macroblock Mode Table)** This field signals which code table is used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer in a P or B picture. This field is identical to the variables MBMODETAB in the VC1 standard, section 9.1.1.33. This field is valid for interlace frame P, B picture and interlace field P, B picture. It is not valid for I picture, nor progressive frame P, B pictures.Two bits are defined for interlace frame P, B pictures; And three bits are defined for interlaced field P, B pictures.Two bits are defined for interlace frame P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to 4-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 3Other encodings are invalidThree bits are defined for interlace field P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to Mixed-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 34 = Code Table 45 = Code Table 56 = Code Table 67 = Code Table 7This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	7:6	**TransAcY (Picture-level Transform Luma AC Coding Set Index, TRANSACTABLE2** BitFieldDesc
	5:4	**TransAcUV (Picture-level Transform Chroma AC Coding Set Index, TRANSACTABLE)** This field, together with PQINDEX, specifies which intra AC coding set to be used for decoding the non-zero AC coefficients in a coded luma (Y) block. This field is the combination of the variables TRANSACFRM and TRANSACFRM2 in the VC1 standard.For I pictures, TransAcY is the same as TRANSACFRM2. For other pictures, it is the same as TRANSACFRM, and therefore must be programmed to be the same as TransAcUV. This field is valid for all picture types.0 = Coding set index 01 = Coding set index 12 = Coding set index 23 is invalidThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	3	**TransDcTab (Intra Transform DC Table)** This field specifies whether the low motion tables or the high motion tables are used to decode the Transform DC coefficients in intra-coded blocks. This field is identical to the variable TRANSDCTAB in the VC1 standard, section 8.1.1.2.This field is valid for all picture types.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	2:0	**CbpTab (Coded Block Pattern Table)** This field specifies the table used to decode the CBPCY syntax element for each coded macroblock in P and B pictures. This field is combination of the variable CBPTAB for P and B frame pictures and the variable ICBPTAB in interlace field P, B pictures and interlace frame P, B pictures in the VC1 standard (Table 52 and Table 102).This field is reserved and MBZ for I or BI pictures as I only has a fixed table.000 = Table 0 (Table 169 for P, B frames or Table 124 otherwise)001 = Table 1 (Table 170 for P, B frames or Table 125 otherwise)010 = Table 2 (Table 171 for P, B frames or Table 126 otherwise)011 = Table 3 (Table 172 for P, B frames or Table 127 otherwise)100 = Table 4 (Table 128 for interlace field/frame P, B pictures)101 = Table 5 (Table 129 for interlace field/frame P, B pictures)110 = Table 6 (Table 130 for interlace field/frame P, B pictures)111 = Table 7 (Table 131 for interlace field/frame P, B pictures)This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		The high motion tables
1h		The low motion tables

MFD_VC1_SHORT_PIC_STATE

MFD_VC1_SHORT_PIC_STATE

Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_SHORT_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:24	**Reserved** Format: MBZ
	23:16	**Picture Height** Format: U8-1 Picture Height in Macroblocks This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).This field is used in VLD and IT modes.Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boudary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.

MFD_VC1_SHORT_PIC_STATE

	15:8	**Reserved** Format: MBZ
	7:0	**Picture Width** Format: U8-1 Picture Width in Macroblocks This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16).This field is used in VLD and IT modes.
2	31:24	**Bitplane Buffer Pitch Minus 1** Format: U7-1 Pitch in Bytes Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format (Gen6 and Gen7), it is written by an application and later read by the HW. In VC1 Long Format (Gen6 and Gen7), it is written by an application, and later read by the HW. But in VC1 Short Format (Gen7 only), it is written and read by H/W only.This field is specified for better performanceFor Gen6 : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Long Format : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row.This field is not used in IT mode, used in VLD mode only.For VC1 DXVA2 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.
	23	**Interpolation Rounder Control** Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process. Note: This bit field is taken from bRcontrol in DXVA_PictureParameters data structure This field is used in VLD and IT modes.
	22:20	**Reserved** Format: MBZ
	19:16	**Motion Vector Mode** This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision.0XX0 = Chroma Quarter -pel + Luma bicubic. (can only be 1MV)0XX1 = Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV)1XX0 = Chroma Quarter -pel + Luma bilinear. (can only be 1MV)1XX1 = Chroma Half-pel + Luma bilinearNote: Bits 19:16 are taken from bMVprecisionAndChromaRelation in DXVA_PictureParameters data structure.Bit 19 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 16 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion.This field is used in both VLD and IT modes.Before the polarity of Chroma Half-pel or Q-pel is reversed from DXVA2 Spec, now I have fixed it to match with DXVA2 VC1 Spec. ???

MFD_VC1_SHORT_PIC_STATE

	15	**DmvSurfaceValid** Indicated when the DMV read surface is valid. This surface stored the direct motion vectors. This field is set fo B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture). This field is not used in IT mode, used in VLD mode only.
	14:12	**Reserved** Format: MBZ
	11	**VC1 Profile** specifies the bitstream profile. Note: This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not. This field is used in both VLD and IT modes. Value / Name / Description 0h / **[Default]** / current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile) 1h / / current picture is in Advanced Profile
	10:6	**Reserved** Format: MBZ
	5	**Backward Prediction Present Flag** Note : a B picture that only uses forward prediction may have this flag set to 1 as well. Driver may still need to provide a valid reference picture index. This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicBackwardPrediction in DXVA2 VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode.
	4	**Intra Picture Flag** This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicIntra in DXVA2 VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode. Value / Name / Description 0h / / entire picture can have a mixture of intra and inter MB type or just inter MB type. 1h / / entire picture is coded in intra MB type
	3	**SecondField** This flag is set for the second field in field pictures.This field is used in both VLD and IT modes.

MFD_VC1_SHORT_PIC_STATE

	2	**Reserved** Format: MBZ
	1:0	**Picture Structure** This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicStructure in DXVA2 VC1 spec. The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in DXVA2 VC1 VLD and IT mode. **Value** / **Name** / **Description** 01b / / top field (bit 0) 10b / / bottom field (bit 1) 11b / / frame (both fields are present) 00b / / illegal
3	31	**Reserved** Format: MBZ
	30	**Overlap Smoothing Enable Flag** This field is the decoded syntax element OVERLAP in bitstreamIndicates if Overlap smoothing is ON at the picture levelThis field is used in both VLD and IT modes **Value** / **Name** / **Description** 0h / Disable / to disable overlap smoothing filter 1h / Enable / to enable overlap smoothing filter
	29	**Range Reduction Scale** Access: None This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled.NOTE: This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent. **Value** / **Name** / **Description** 0h / Disable **[Default]** / Scale down reference picture by factor of 2 1h / Enable / Scale up reference picture by factor of 2
	28	**Range Reduction Enable** This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element (DXVA_PictureParameters bPicDeblocked bit 5) in the Picture Header.This field is for

MFD_VC1_SHORT_PIC_STATE

Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture.RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description
0h	Disable **[Default]**	Range reduction is not performed
1h	Enable	Range reduction is performed

27:24 **Reserved**

Format:	MBZ

23:22 **Progressive Pic Type**

This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicExtrapolation in DXVA2 VC1 spec.The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in DXVA2 VC1 VLD and IT mode.

Value	Name	Description
0		progressive only picture
1		progressive only picture
2		interlace picture (frame-interlace or field-interlace)
3		illegal

21 **Reserved**

Format:	MBZ

20:16 **P-Pic Ref Distance**

Access:	None

This element defines the number of frames between the current frame and the reference frame. It is the same as the REFDIST SE in VC1 interlaced field picture header. It is present if the entry-level flag REFDIST_FLAG == 1, and if the picture type is not one of the following types: B/B, B/BI, BI/B, BI/BI. If the entry level flag REFDIST_FLAG == 0, REFDIST shall be set to the default value of 0.This field is used in DXVA2 VC1 VLD mode only, not used in IT and intel VC1 VLD Long Format modes.

Value	Name
0-16	unsigned integer
0h	**[Default]**

MFD_VC1_SHORT_PIC_STATE

15:14 QUANTIZER

Value	Name	Description
00b		implicit quantizer at frame leve
01b		explicit quantizer at frame level, and use PQUANTIZER SE to specify uniform or non-uniform
10b		explicit quantizer, and non-uniform quantizer for all frames
11b		explicit quantizer, and uniform quantizer for all frames

13 MULTIRES Present Flag (for Simple/Main Profile only)

Value	Name	Description
0h		RESPIC Parameter is present in the picture header
1h		RESPIC Parameter is present in the picture header

12 SYNCMARKER Present Flag (for Simple/Main Profile only)

Value	Name	Description
0		Bitstream for Simple and Main Profile has no sync marker
1		Bitstream for Simple and Main Profile may have sync marker(s)

11 RANGERED Present Flag (for Simple/Main Profile only)

It is needed for Picture Header Parsing.Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description
0		Range Reduction Parameter (RANGEREDFRM) is not present in the picture header
1		Range Reduction Parameter (RANGEREDFRM) is present in the picture header.

10:8 MAXBFRAMES

Number of consecutive B Frames.

7 PANSCAN Present Flag

Value	Name	Description
0		Pan Scan Parameters are not present in the picture header
1		Pan Scan Parameters are present in the picture header

6 REFDIST_FLAG

For header parsing REFDIST.This is used in DXVA2 VC1 VLD mode only, not used in IT and intel VC1 VLD modes.

5 LOOPFILTER Enable Flag

This filed is the decoded syntax element LOOPFILTER in bitstream. It indicates if In-loop Deblocking is ON according to picture level bitstream syntax control. This bit affects BSD unit and also the loop filter unit.When this bit is set to 1, PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command must also be set to 1. In this case, in-loop deblocking

MFD_VC1_SHORT_PIC_STATE

operation follows the VC1 standard - deblocking doesn't cross slice boundary.When this bit is set to 0, but PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command is set to 1. It indicates the loop filter unit is used for out-of-loop deblocking. In this case, deblocking operation does cross slice boundary.This field is used in VLD mode only, not in IT mode.

Value	Name	Description
0		In-Loop-Deblocking-Filter is disabled
1		In-Loop-Deblocking-Filter is enabled

4

FastUVMCFlag (Fast UV Motion Compensation Flag)

This field specifies whether the motion vectors for UV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard.This field is used in both VLD and IT modes.It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector Mode LSBit.

Value	Name	Description
0h		no rounding
1h		quarter-pel offsets to half/full pel positions

3

EXTENDED_MV Present Flag

BitFieldDesc

Value	Name	Description
0h		Extended_MV is not present in the picture header
1h		Extended_MV is present in the picture header

2:1

DQUANT

Access:	None
Format:	U2

Use for Picture Header Parsing of VOPDUANT elements

Value	Name	Description
0h	**[Default]**	
00b		no VOPDQUANT elements; Quantizer cannot vary in frame, same quantization step size PQUANT is used for all MBs in the frame
01b		refer to VC1 Spec. for all the MB position dependent quantizer selection
10b		The macroblocks located on the picture edge boundary shall be quantized with ALTPQUANT while the rest of the macroblocks shall be quantized with PQUANT.
11b	Reserved	

MFD_VC1_SHORT_PIC_STATE

	0	**VSTRANSFORM flag** Value / Name / Description: 0h / Disable / variable-sized transform coding is not enabled 1h / Enable / variable-sized transform coding is enabled
4	31:29	**Reserved** Format: MBZ (for possible future change to BFraction Enumeration)
	28:24	**BFraction Enumeration** This field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1 standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21 encodings of BFRACTION as shown in the table here. Other values are reserved. The VLD decoded value of BFRACTION (from the picture header) is mapped into an enum value from 0 to 20.(??? MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine Motion Prediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "ScaleFactor >= 128". ??? How can the enum replicate this feature ???)This field is only valid for B pictures. This field is used only in DXVA2 VC1 VLD mode, it is not used in Intel VC1 VLD Long Format mode and IT mode.BFRACTION VLCBFRACTION Enum0001/200011/310102/320111/431003/441011/551102/5611100003/5711100014/5811100101/6911100115/61011101001/71111101012/71211101103/71311101114/71411110005/71511110016/71611110101/81711110113/81811111005/81911111017/8201111111BI Pic Indicator31 (optional)
	23	**Reserved** Format: MBZ Advanced Profile only; RANGE_MAPY_FLAG Range Mapping not supported
	22:20	**Reserved** Format: MBZ Advanced Profile only; RANGE_MAPY Range Mapping not supported
	19	**Reserved** Format: MBZ Advanced Profile only; RANGE_MAPUV_FLAG Range Mapping not supported
	18:16	**Reserved** Format: MBZ Advanced Profile only; RANGE_MAPUV Range Mapping not supported
	15:9	**Reserved** Format: MBZ
	8	**4MV Allowed Flag**
	7	**POSTPROC Flag**
	6	**PULLDOWN**
	5	**INTERLACE**
	4	**TFCNTRFLAG**
	3	**FINTERFLAG**

MFD_VC1_SHORT_PIC_STATE

	2	**REFPIC Flag** For a BI picture, REFPIC flag must set to 0For I and P picture, REFPIC flag must set to 0.For a B picture, REFPIC flag must set to 0, except for a B-field in interlaced field mode which can be 0 or 1 (e.g. the top B field can be used as a reference for decoding its corresponding bottom B-field in a field pair).In VLD mode, this flag cannot be used as an optimization signaling for an I or P picture that is not used as a reference picture.This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicDeblockConfined[bit2] in DXVA2 VC1 spec.The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode. **Value** / **Name** / **Description** 0h — the current picture after decoded, will never used as a reference picture 1h — the current picture after decoded, will be used as a reference picture later
	1	**PSF**
	0	**EXTENDED_DMV Present Flag** **Value** / **Name** / **Description** 0h — **[Default]** — Extended_DMV is not present in the picture header 1h — Extended_DMV is present in the picture header

MFD_VP8_BSD_OBJECT

MFD_VP8_BSD_OBJECT

Source: VideoCS

Length Bias: 2

The MFD_VP8_BSD_OBJECT command is the only primitive command for the VP8 Decoding Pipeline. The Partitions of the bitstream is loaded as indirect data object.

Before issuing a MFD_VP8_BSD_OBJECT command, all VP8 frame level states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VP8_BSD_OBJECT command.

Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VP8_BSD_OBJECT Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8_DEC Format: OpCode
	23:21	**subOpcodeA** Default Value: 1h Format: OpCode
	20:16	**subOpcodeB** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 14h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20:16	**Partition0 CPBAC Entropy Count** Pass the Partition0 CPBAC State to HW. Max value is 24.

MFD_VP8_BSD_OBJECT

	15:8	**Partition0 CPBAC Entropy Range** Pass the Partition0 CPBAC State to HW.
	7:6	**Reserved** Format: MBZ
	5:4	**Coded Num of Coeff Token Partitions** Num of Partitions = 2^CodedNumCoeffTokenParititons. 0 = 1 Partition only 1 = 2 Partitions 2 = 4 Partitions 3 = 8 Partitions are present in the bitstream.
	3	**Reserved** Format: MBZ
	2:0	**Partition0 FirstMBBitOffset from Frame Header** Allow HW to jump to the location in the bitstream where per MB information starts in the Partition0.
2	31:24	**Partition0 CPBAC Entropy Value** Pass the Partition0 CPBAC State to HW.
	23:0	**Reserved** Format: MBZ
3	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition0 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 0 length + 1) in bytes
4	31:0	**Indirect Partition0 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
5	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition1 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends.

MFD_VP8_BSD_OBJECT

		It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 1 length + 1) in bytes
6	31:0	**Indirect Partition1 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
7	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition2 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 2 length + 1) in bytes
8	31:0	**Indirect Partition2 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
9	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition3 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 3 length + 1) in bytes
10	31:0	**Indirect Partition3 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.

MFD_VP8_BSD_OBJECT

11	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition4 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 4 length + 1) in bytes
12	31:0	**Indirect Partition4 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
13	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition5 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 5 length + 1) in bytes
14	31:0	**Indirect Partition5 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
15	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition6 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 6 length + 1) in bytes

MFD_VP8_BSD_OBJECT

16	31:0	**Indirect Partition6 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
17	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition7 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 7 length + 1) in bytes
18	31:0	**Indirect Partition7 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
19	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition8 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 8 length + 1) in bytes
20	31:0	**Indirect Partition8 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
21	31	**Concealment Method** This field specifies the method used for concealment when error is detected. **Value** / **Name** / **Description** 0 / Intra 16x16 Prediction / A copy from the current picture is performed using Intra 16x16 Prediction method. 1 / Inter P Copy / A copy from collocated macroblock location is performed from the

MFD_VP8_BSD_OBJECT

		concealment reference indicated by the ConCeal_Pic_Id field.

30:18 Reserved

Format:	MBZ

17:16 Conceal_Pic_Id (Concealment Picture ID)

Exists If:	[Concealment Method] == 1

This field identifies the picture in the reference list to be used for concealment. This field is only valid if Concealment Method is Inter P Copy.
00 - Last Decoded Picture
01 - Golden Reference Picture
02 - Alternate Reference Picture
03 - User provided Reference Picture

15 Reserved

Format:	MBZ

14 BSDPrematureComplete Error Handling

It occurs in situation where the decode is completed but there are still data in the bitstream.

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)??

13 Reserved

Format:	MBZ

12 MPR Error (MV out of range) Handling

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)??

11 Reserved

Format:	MBZ

10 Entropy Error Handling

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)??

9 Reserved

Format:	MBZ

MFD_VP8_BSD_OBJECT

	8	**MB Header Error Handling** Value: 0 — Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling Value: 1 — Set the interrupt to the driver (provide MMIO registers for MB address R/W)??
	7:0	**Reserved** Format: MBZ

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_DIRECTMODE_STATE

Source: VideoCS

Length Bias: 2

This is a picture level command and is issued once per picture. All DMV buffers are treated as standard media surfaces, in which the lower 6 bits are used for conveying surface states.Current Pic POC number is assumed to be available in POCList[32 and 33] of the MFX_AVC_DIRECTMODE_STATE Command.This command is only valid in the AVC decoding in VLD and IT modes, and AVC encoder mode. The same command supports both Long and Short DXVA2 AVC Interface. The DMV buffers are not required to be programmed for encoder mode.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_SINGLE_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h Format: OpCode
	20:16	**SubOpcodeB** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0045h Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:6	**Direct MV Buffer Base Address for Picture 0 - Read/Write** Format: GraphicsAddress63-6 Note: This field is changed to one per frame (both top and bottom field share the same Direct MV Buffer Base Address). This field provides the address of the DMV write buffer to store motion vectors decoded in the current picture (top field), which may be used later as a collocated motion information read

MFX_AVC_DIRECTMODE_STATE

		buffer of the associated reference picture in decoding subsequent B-pictures that have MB coded in direct mode. It is a private buffer used by the MPR hardware only. Itscontent is not accessed by software. This buffer must be 64-byte cacheline aligned. The write buffer size is 557,056 bytes for 1 frame. Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution) It is only valid if the current picture is a progressive frame, MbAff frame, or a top field. There are a total of 32 reference picture (previously decoded) Direct MV Buffers (0 to 31, not including the DMV write buffer 32 and 33 of the current picture) to read in the corresponding collocated DMV and motion information. For reference picture, these 32 DMV read Buffers can be indexed by the frame_store_ID[4:0], which is obtained from RefPicList L0/L1[RefPicIdx]. frame_Store_IDbit[0] (indicator for Top/Bottiom Field). For writing out motion information during the decoding of the current picture, all 34 DMV buffers can be addressed by [img_dec_fs_idc[4:0]«1 + img_structure[1]].
	5:0	**Reserved** Format: MBZ
3..32	63:6	**Direct MV Buffer Base Address for Reference Frame 1 to 15 (In Frame)** Format: GraphicsAddress63-6 This field is for the Pre-Deblocking Destination Address, and provides the base address of the DMV buffer for reference frames 2 to 31. They are needed if the current B-Picture has MBs coded in direct mode. This is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned. There are a total of 32 possible Direct MV Read Buffers (not including the current write buffer of the current picture) to read in the corresponding DMV. Each read buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). The adjacent DMV buffers are paired ([2 and 3], [4 and 5], [N and N+1], ..[30 and 31]). This field is changed to one per frame: both top and bottom field share the same Direct MV Buffer Base Address. **Programming Notes** This field is ignored if PreDeblockOutEnable is set to 0 (disable).
	5:0	**Reserved** Format: MBZ
33	31:15	**Reserved** Format: MBZ
	14:13	**Direct MV Buffer Base Address for Reference Frame - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources

MFX_AVC_DIRECTMODE_STATE

3h	Reserved	

12:11 Reserved

Format:	MBZ

10 Direct MV Buffer Base Address for Reference Frame - Memory Compression Mode

Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details.

Note: This bit is not used unless Memory Compression Enable is set to "1"

Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

9 Direct MV Buffer Base Address for Reference Frame - Memory Compression Enable

Value	Name
0	Compression Disable

Programming Notes
This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed

8:7 Direct MV Buffer Base Address for Reference Frame - Arbitration Priority Control

Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:1 Direct MV Buffer Base Address for Reference Frame - Index to Memory Object Control State (MOCS) Tables

Format:	U6

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers.

The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 Reserved

MFX_AVC_DIRECTMODE_STATE

34..35	63:6	**Direct MV Buffer Base Address for Write (Write-Only Buffer)(in frame)** Format: GraphicsAddress63-6 This field provides the base address of the DMV write-only buffer for the current decoding frame/field. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned, i.e. the same as the above DMV read/write buffers. These 2 buffers can only be addressed by [img_dec_fs_idc[4:0]«1 + img_structure[1]] for the current picture being decoded. Each write buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). DMV write buffer 32 is valid only if the current picture is a progressive frame, MbAff frame, or a top field. DMV write buffer 33 is valid only if the current picture is a bottom field. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example, GraphicsAddress[47:12] is a 4KB page address.
	5:0	**Reserved** Format: MBZ
36	31:15	**Reserved** Format: MBZ
	14:13	**Direct MV Buffer Base Address for Write - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Direct MV Buffer Base Address for Write - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

MFX_AVC_DIRECTMODE_STATE

	9	**Direct MV Buffer Base Address for Write - Memory Compression Enable** Value: 0 — Name: Compression Disable **Programming Notes** This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Direct MV Buffer Base Address for Write - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b — Name: Highest priority Value: 01b — Name: Second highest priority Value: 10b — Name: Third highest priority Value: 11b — Name: Lowest priority
	6:1	**Direct MV Buffer Base Address for Write - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
37..70	31:0	**POC List, POCList[34][31:0]** Each POC value is a signed 32-bit number. One-to-one correspondence with the 34 Direct MV Buffer Address for Reference and Currrent Frames/Fields There are 34 POC entries in the list. For reference picture, only the lower 32 POC [0-31] entries can be used, and POCList[ ] is indexed by the frame_store_ID[4:0], which is obtained from RefPicList L0/L1[RefPicIdx]. frame_Store_IDbit[0] (indicator for Top/Bottiom Field). For current picture, all 34 POC entries [0-33] can be addressed by POCList[img_dec_fs_idc[4:0]«1 + img_structure[1]]. For frame-only mode, every other entry is skipped. For MBAFF and field-only picture, each entry is a field POC, and every two entries are paired.

MFX_AVC_IMG_STATE

MFX_AVC_IMG_STATE

Source: VideoCS

Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands. This command supports both Long and Short VLD and IT DXVA2 AVC Decoding Interface.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_IMG_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0Ch Excludes DWord (0,1) Format: =n 00Eh, used for normal decode and encode mode000h, a special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware.
1	31:16	**Reserved** Format: MBZ

MFX_AVC_IMG_STATE

	15:0	**Frame Size** Format: U16 in MB unit This field is only valid for encoder. Its ignored for decoder.The value for FrameSizeInMBs must match the product of FrameWidthInMBs and FrameHeightInMBs. For encode mode if frame size is 4Kx4K, then we need to specify 65535 in this field as we dont have enough bits to specify 65536. This is used only for cabac zero work insertion when in encode mode. Value: [1,65535] — Name: — Description: representing Number of MBs [1,65535]
2	31:24	**Reserved** Format: MBZ (bit[31:24] must be zero to match the DXVA 16-bit definition for FrameHeightInMBsMinus1)
	23:16	**Frame Height** Format: U8-1 in MB unit It is set to the value of (FrameHeightInMBsMinus1+ 1). Since the max value for FrameHeightInMBs is 255, the max allowed value for FrameHeightInMBsMinus1 is only 254. The min value for FrameHeightInMBs is 1.Although the max. value that can be specified for FrameHeightInMBs is 255 (in the current implementation), FrameWidthInMBs * FrameHeightInMBs must not exceed the max value of FrameSizeInMBs[14:0].e.g. for 1920x1080, FrameHeightInMBs[7:0] is equal to 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).It is derived from FrameHeightInMbs = (2 - frame_mbs_only_flag) * PicHeightInMapUnits and PicHeightInMbs = FrameHeightInMbs / (1 + field_pic_flag) internally done. For MBAFF, PicHeightInMapUnits is in MB pair unit, so the bitstream sends only half frame height. Value: [0,255] — Name: — Description: representing height [1,256]
	15:8	**Reserved** Format: MBZ (bit[15:8] must be zero to match the DXVA 16-bit definition for FrameWidthInMBsMinus1)
	7:0	**Frame Width** Format: U8-1 in MB unit It is set to the value of (FrameWidthInMBsMinus1+ 1). Since the max value for FrameWidthInMBs is 255, the max allowed value for FrameWidthInMBsMinus1 is only 254. The min value for FrameWidthInMBs is 1.Although the max. value that can be specified for FrameWidthInMBs is 255 (in the current implementation), FrameWidthInMBs * FrameWidthInMBs must not exceed the max value of FrameSizeInMBs[14:0].e.g. for 1920x1080, FrameHeightInMBs[7:0] is equal to 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).It is derived from FrameWidthInMbs = (2 - frame_mbs_only_flag) * PicWidthInMapUnits and

MFX_AVC_IMG_STATE

PicWidthInMbs = FrameWidthInMbs / (1 + field_pic_flag) internally done. For MBAFF, PicWidthInMapUnits is in MB pair unit, so the bitstream sends only half frame width.

Value	Name	Description
[0,255]		representing width [1,256]

3 — 31:29 Reserved

Format:	MBZ

(bit[31:29] must be zero to match the DXVA2 8-bit definition for InitQpChroma[1])

28:24 Second Chroma QP Offset

Signed integer value. It should be in the range of -12 to +12 (according to AVC spec).It specifies the offset for determining QP Cr from QP Y. It is set to the upper 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS.Chroma_qp_offset [4:0] - chroma_qp_offset_bits (from the current active PPS)Chroma_qp_offset [9:5] - second_chroma_qp_offset_bits

23:21 Reserved

Format:	MBZ

(bit[23:21] must be zero to match the DXVA2 8-bit definition for InitQpChroma[1])

20:16 First Chroma QP Offset

Signed integer value. It should be in the range of -12 to +12 (according to AVC spec).It specifies the offset for determining QP Cb from QP Y. It is set to the lower 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS.Chroma_qp_offset [4:0] - chroma_qp_offset_bits (from the current active PPS)Chroma_qp_offset [9:5] - second_chroma_qp_offset_bits

15:14 Reserved

Format:	MBZ

13 RhoDomain Rate Control Enable

Format:	Enable

This field indicates if RhoDomain related parameters are present in the MFX_AVC_IMAGE_STATE. (AverageMacroblockQP). It enables the Rho Domain statistics collection.

Value	Name	Description
0	Disable	RhoDomain rate control parameters are not present in MFX_AVC_IMAGE_STATE
1	Enable	RhoDomain rate control parameters are present in MFX_AVC_IMAGE_STATE.

Programming Notes
This field must set to '0' for B pictures.

12 Weighted_Pred_Flag

Format:	Enable

(This field is defined differently from Gen6, Gen7 follows strictly DXVA2 AVC interface.)

MFX_AVC_IMG_STATE

Value	Name	Description
0	Disable **[Default]**	specifies that weighted prediction is not used for P and SP slices
1	Enable	specifies that weighted prediction is used for P and SP slices

Programming Notes
This field must set to '0' for B and I pictures.

11:10 **Weighted_BiPred_Idc**

(This field follows strictly DXVA2 AVC interface.)

Value	Name	Description
0	DEFAULT **[Default]**	Specifies that the default weighted prediction is used for B slices
1	EXPLICIT	Specifies that explicit weighted prediction is used for B slices
2	IMPLICIT	Specifies that implicit weighted prediction is used for B slices.
3	Reserved	Illegal value

Programming Notes
This field must set to 0 for P and I pictures.

9:8 **ImgStruct - Image Structure, img_structure[1:0]**

The current encoding picture structure can only takes on 3 possible values

Value	Name
00b	Frame Picture
01b	Top Field Picture
11b	Bottom Field Picture
10b	Invalid, not allowed.

Programming Notes
img_structure[0] can be used as a flag to distinguish between frame and field structure. It must be consistent with the field_pic_flag setting in the Slice Header.This parameter is specified for Intel interface only, not present in the DXVA as a separate state (instead the img_structure[1] is embedded inside the DXVA picture definition).

7:0 **Reserved**

Format:	MBZ

4 31:16 **MinFrameWSize**

Default Value:	0h
Format:	U16

MFX_AVC_IMG_STATE

Minimum Frame Size [15:0] (in Word, 16-bit)(Encoder Only) Mininum Frame Size is specified to compensate for intel Rate Control Currently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Mininum Frame Size, is always less than maximum frame size **FrameBitRateMax (DWORD 10 bits** 29:16).This field is reserved in Decode mode.
The programmable range 0...2^18-1
When MinFrameWSizeUnits is 00.
Programmable range is 0...2^20-1 when MinFrameWSizeUnits is 01.
Programmable range is 0...2^26-1 when MinFrameWSizeUnits is 10.
Programmable range is 0...2^32-1 when MinFrameWSizeUnits is 11.

15 **MbStatEnabled**

Format:	Enable

Enable reading in MB status buffer (a.k.a. encoding stream-out buffer) Note: For multi-pass encoder, all passes except the first one need to set this value to 1. By setting the first pass to 0, it does save some memory bandwidth.

Value	Name	Description
0	Disable	Disable Reading of Macroblock Status Buffer
1	Enable	Enable Reading of Macroblock Status Buffer

14 **LoadSlicePointerFlag**

Format:	Enable

LoadBitStreamPointerPerSlice (Encoder-only)To support multiple slice picture and additional header/data insertion before and after an encoded slice.When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream.When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0	Disable	Load BitStream Pointer only once for the first slice of a frame
1	Enable	Load/reload BitStream Pointer only once for the each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field

13 **Reserved**

12 **MvUnpackedFlag**
MVUnPackedEnable (Encoder Only)This field is reserved in Decode mode.

Value	Name	Description
0	PACKED	use packed MV format (compliant to DXVA)
1	UNPACKED	use unpacked 8MV/32MV format only

MFX_AVC_IMG_STATE

11:10 **ChromaFormatIdc**

Chroma Format IDC, ChromaFormatIdc[1:0]It specifies the sampling of chroma component (Cb, Cr) in the current picture as follows :

Value	Name	Description
00b	monochrome picture	Desc
01b	4:2:0 picture	Desc
10b	4:2:2 picture (not supported)	
11b	4:4:4 picture (not supported)	

Programming Notes
It is set to the value of the syntax element read from the current active SPS.The corresponding Monochrome Flag (monochrome_flag) can be derived from this field.

9 **Reserved**

Format:	MBZ

8 **MbMvFormatFlag**

Use MB level MvFormat flag (Encoder Only)

Value	Name	Description
0	IGNORE	HW PAK ignore MvFormat in the MB data. When bit 12 == 0, all MBs use packed MV formatWhen bit 12 == 1, each MB data must use unpacked MV format, 8MV when there is no minor MV involved, and 32MV if there are some minor MVs.
1	FOLLOW	HW PAK will follow MvFormat value set within each MB data.

Programming Notes
They must take one of the two values: the 8MV unpacked format (MvFormat =101b), and the 32MV unpacked format (MvFormat =110b).This bit can be set only when MvUnpackedFlag (bit 12 of this register) is set otherwise system could hang.

7 **EntropyCodingFlag**

Entropy Coding Flag, entropy_coding_flag

Value	Name	Description
0	CAVLC bit-serial encoding mode	Desc
1	CABAC bit-serial encoding mode.	Desc

Programming Notes
It specifies one of the two possible bit stream encoding modes in the AVC. It is set to the value of the syntax element read from the current active PPS.

MFX_AVC_IMG_STATE

6 **ImgDisposableFlag**

Current Img Disposable Flag or Non-Reference Picture Flag

Value	Name	Description
0	REFERENCE	the current decoding picture may be used as a reference picture for others
1	DISPOSABLE	the current decoding picture is not used as a reference picture (e.g. a B-picture cannot be a reference picture for any subsequent decoding)

Programming Notes
It is derived from ImgDisposableFlag = (nal_ref_idc == 0). nal_ref_idc is a syntax element from a NAL unit. When this flag is set, no reference picture and DMV are written out.This field is only valid for VLD decoding mode.

5 **ConstrainedIPredFlag**

Constrained Intra Prediction Flag, constrained_ipred_flagIt is set to the value of the syntax element in the current active PPS.

Value	Name	Description
0	INTRA_AND_INTER	allows both intra and inter neighboring MB to be used in the intra-prediction encoding of the current MB.
1	INTRA_ONLY	allows only to use neighboring Intra MBs in the intra-prediction encoding of the current MB. If the neighbor is an inter MB, it is considered as not available.

4 **Direct8x8InfFlag**

Direct 8x8 Inference Flag, direct_8x8_inference_flagIt is set to the value of the syntax element in the current active SPS.It specifies the derivation process for luma motion vectors in the Direct MV coding modes (B_Skip, B_Direct_16x16 and B_Direct_8x8). When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.It must be consistent with the frame_mbs_only_flag and transform_8x8_mode_flag settings.

Value	Name	Description
0	SUBBLOCK	allows subpartitioning to go below 8x8 block size (i.e. 4x4, 8x4 or 4x8)
1	BLOCK	allows processing only at 8x8 block size. MB Info is stored for 8x8 block size.

3 **Transform8x8Flag**

8x8 IDCT Transform Mode Flag, trans8x8_mode_flagSpecifies 8x8 IDCT transform may be used in this pictureIt is set to the value of the syntax element in the current active PPS.

Value	Name	Description
0	4x4	no 8x8 IDCT Transform, only 4x4 IDCT transform blocks are present
1	8x8	8x8 Transform is allowed

MFX_AVC_IMG_STATE

	2	**FrameMbOnlyFlag** Frame MB only flag, frame_mbs_only_flagIt is set to the value of the syntax element in the current active SPS. **Value** / **Name** / **Description** 0 / FALSE / not true ; effectively enables the possibility of MBAFF mode. 1 / TRUE / true, only frame MBs can occur in this sequence, hence disallows the MBAFF mode and field picture.
	1	**MbaffFlameFlag** MBAFF mode is active, mbaff_frame_flag.It is derived from MbaffFrameFlag = (mb_adaptive_frame_field_flag && ! field_pic_flag). mb_adaptive_frame_field_flag is a syntax element in the current active SPS and field_pic_flag is a syntax element in the current Slice Header. They both are present only if frame_mbs_only_flag is 0. Although mbaff_frame_flag is a Slice Header parameter, its value is expected to be the same for all the slices of a picture.It must be consistent with the mb_adaptive_frame_field_flag, the field_pic_flag and the frame_mbs_only_flag settings.This bit is valid only when the img_structure[1:0] indicates the current picture is a frame. **Value** / **Name** / **Description** 0 / FALSE / not in MBAFF mode 1 / TRUE / in MBAFF mode
	0	**FieldPicFlag** Field picture flag, field_pic_flag, specifies the current slice is a coded field or not.It is set to the same value as the syntax element in the Slice Header. It must be consistent with the img_structure[1:0] and the frame_mbs_only_flag settings.Although field_pic_flag is a Slice Header parameter, its value is expected to be the same for all the slices of a picture. **Value** / **Name** / **Description** 0h / FRAME / a slice of a coded frame 1h / FIELD / a slice of a coded field
5 [ExistsIf]Encode Only	31	**Trellis Quantization Enabled (TQEnb)** Format: / Enable The TQ improves output video quality of AVC CABAC encoder by selecting quantized values for each non-zero coefficient so as to minimize the total R-D cost.This flag is only valid AVC CABAC mode. Otherwise, this flag should be disabled. **Value** / **Name** / **Description** 0h / Disable / Use Normal 1h / Enable / Use Trellis quantization

MFX_AVC_IMG_STATE

30:28 **Trellis Quantization Rounding (TQR)**

This rounding scheme is only applied to the quantized coefficients ranging from 0 to 1 when TQEnb is set to 1 in AVC CABAC mode. One of the following values is added to quantized coefficients before truncating fractional part.

Value	Name	Description
000b		Add 1/8
001b		Add 2/8
010b		Add 3/8
011b		Add 4/8 (rounding 0.5)
100b		Add 5/8
101b		Add 6/8
110b	Default	Add 7/8 (Default rounding 0.875)

27 **Trellis Quantization Chroma Disable (TQChromaDisable)**

This signal is used to disable chroma TQ. To enable TQ for both luma and chroma, TQEnb=1, TQChromaDisable=0. To enable TQ only for luma, TQEnb=1, TQChromaDisable=1.

Value	Name	Description
0h		Enable Trellis Quantization chroma
1h	Default	Disable Trellis Quantization chroma

26:17 **Reserved**

Format:	MBZ

16 **NonFirstPassFlag**

This signals the current pass is not the first pass. It will imply designate HW behavior: e.g

Value	Name	Description
0h	Disable	Always use the MbQpY from initial PAK inline object for all passes of PAK
1h	Enable	Use MbQpY from stream-out buffer if MbRateCtrlFlag is set to 1

15:13 **Reserved**

Format:	MBZ

12 **Reserved**

Format:	MBZ

11:10 **MinFrameWSizeUnits**

This field is the Minimum Frame Size Units

Value	Name	Description
00b	compatibility mode	Minimum Frame Size is in old mode (words, 2bytes)
01b	16 byte	Minimum Frame Size is in 16bytes
10b	4Kb	Minimum Frame Size is in 4Kbytes

MFX_AVC_IMG_STATE

11b	16Kb	Minimum Frame Size is in 16Kbytes

9 — MbRateCtrlFlag - MB level Rate Control Enabling Flag

MB Rate Control conformance mask

Value	Name	Description
0h	Disable	Apply accumulative delta QP for consecutive passes on top of the macroblock QP values in inline data
1h	Enable	Apply RC QP delta to suggested QP values in Macroblock Status Buffer except the first pass.

Programming Notes
This field is ignored when MacroblockStatEnable is disabled or MB level Rate control flag for the current MB is disable in Macroblock Status Buffer.

8 — Reserved

Format:	MBZ

7 — Intra/InterMbIpcmFlag - ForceIPCMControlMask

This field is to Force **IPCM** for Intra or Inter Macroblock size conformance mask.

Value	Name	Description
0h	Disable	Do not change intra or Inter macroblocks even
1h	Enable	Change intra or Inter macroblocks MB_type to IPCM

Programming Notes
This field is ignored when MacroblockStatEnable is disabled or MB level Intra MB conformance flag for the current MB is disable in Macroblock Status Buffer.

6:4 — Reserved

Format:	MBZ

3 — FrameSzUnderFlag - FrameBitRateMinReportMask

This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin

Value	Name	Description
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1h	Enable	set bit0 and bit 1of MFC_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit rate Minimum limit.

2 — FrameSzOverFlag - FrameBitRateMaxReportMask

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.

MFX_AVC_IMG_STATE

		1 | Enable | Set bit0 and bit 1 of MFC_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit rate Maximum limit.
	1	**InterMbMaxBitFlag - InterMBMaxSizeReportMask** This is a mask bit controlling if the condition of any inter MB in the frame exceeds InterMBMaxSize. Value | Name | Description 0 | Disable | Do not update bit0 of MFC_IMAGE_STATUS control register. 1 | Enable | Set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Inter MB Conformance Max size limit.
	0	**IntraMbMaxBitFlag - IntraMBMaxSizeReportMask** This is a mask bit controlling if the condition of any intra MB in the frame exceeds IntraMBMaxSize. Value | Name | Description 0h | Disable | Do not update bit0 of MFC_IMAGE_STATUS control register. 1 | Enable | set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Intra MB Conformance Max size limit.
6 [ExistsIf]Encode Only	31:28	**Reserved**
	27:16	**InterMbMaxSz** Format: | U12 This field, Inter MB Conformance Max size limit,indicates the allowed max bit count size for Inter MB
	15:12	**Reserved** Format: | MBZ
	11:0	**IntraMbMaxSz** Exists If: | //Intra Only Format: | U12 This field, Intra MB Conformance Max size limit,indicates the allowed max bit count size for Intra MB All IPCM MBs should ignore this Max size limit.
7 [ExistsIf]Encode Only	31:17	**Reserved**
	16	**Reserved**
	15:1	**Reserved**

MFX_AVC_IMG_STATE

	0	**VSL Top MB Trans8x8flag** Value / Name / Description 0 / Disable **[Default]** / VSL will only fetch the current MB data. 1 / Enable / When this bit is set VSL will make extra fetch to memory to fetch the MB data for top MB.
8 [ExistsIf]Encode Only	31:24	**SliceDeltaQpMax[3]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for total bit-count above FrameBitRateMax - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta»3).
	23:16	**SliceDeltaQpMax[2]** Format: U8 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/8 and below 1/ 4 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and ¼ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»3), (FrameBitRateMax+ FrameBitRateMaxDelta»2).
	15:8	**SliceDeltaQpMax[1]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above1/ 4 and below 1/2 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between ¼ and ½ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»2), (FrameBitRateMax+ FrameBitRateMaxDelta»1).
	7:0	**SliceDeltaQpPMax[0]** Format: S7 Range: [0:MAX_QP_DELTA]

MFX_AVC_IMG_STATE

		This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta , i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»1), infinite).
9 [ExistsIf]Encode Only	31:24	**SliceDeltaQpMin[3]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for total bit-count below FrameBitRateMin - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»3), FrameBitRateMin).
	23:16	**SliceDeltaQpMin[2]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count below FrameBitRateMin - below 1/ 8 and above 1/ 4This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»2), (FrameBitRateMin- FrameBitRateMinDelta»3)).
	15:8	**SliceDeltaQpMin[1]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count below FrameBitRateMin- below 1/4 and above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»1), (FrameBitRateMin- FrameBitRateMinDelta»2)).
	7:0	**SliceDeltaQpMin[0]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice Level Delta QP for bit-count below FrameBitRateMin - below 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS

MFX_AVC_IMG_STATE

<table>
<tr><td></td><td></td><td>control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta , i.e., in the range of [0, (FrameBitRateMin- FrameBitRateMinDelta»1).</td></tr>
<tr><td rowspan="4">10
[ExistsIf]Encode Only</td><td>31</td><td>FrameBitrateMaxUnit
This field is the Frame Bitrate Maximum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Byte</td><td>FrameBitRateMax is in units of 32 Bytes when FrameBitrateMaxUnitMode is 1 and in units of 128 Bytes if FrameBitrateMaxUnitMode is 0</td></tr>
<tr><td>1</td><td>Kilo Byte</td><td>FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0</td></tr>
</table></td></tr>
<tr><td>30</td><td>FrameBitrateMaxUnitMode
This field is the Frame Bitrate Maximum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>compatibility mode</td><td>FrameBitRateMaxUnit is in old mode (128b/16Kb)</td></tr>
<tr><td>1h</td><td>New mode</td><td>FrameBitRateMaxUnit is in new mode (32byte/4Kb)</td></tr>
</table></td></tr>
<tr><td>29:16</td><td>FrameBitRateMax
This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28 and 29 should be 0..
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0-512KB</td><td></td><td>The programmable range is 0-512KB when FrameBitrateMaxUnit is 0.</td></tr>
<tr><td>0-8190KB</td><td></td><td>The programmable range is 0-8190KB when FrameBitrateMaxUnit is 1.</td></tr>
</table></td></tr>
<tr><td>15</td><td>FrameBitrateMinUnit
This field is the Frame Bitrate Minimum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Byte</td><td>FrameBitRateMax is in units of 32 Bytes when FrameBitrateMinUnitMode is 1 and in units of 128 Bytes if FrameBitrateMinUnitMode is 0</td></tr>
<tr><td>1</td><td>Kilo Byte</td><td>FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0</td></tr>
</table></td></tr>
</table>

MFX_AVC_IMG_STATE

	14	**FrameBitrateMinUnitMode** This field is the Frame Bitrate Minimum Limit Units. **Value** / **Name** / **Description** 0h / Compatibility mode / FrameBitRateMaxUnit is in old mode (128b/16Kb) 1h / New mode / FrameBitRateMaxUnit is in new mode (32byte/4Kb)
	13:0	**FrameBitRateMin** RangeThe programmable range 0-512KB When FrameBitrateMinUnit is in 0.Programmable range is 0-8190 KB when FrameBitrateMinUnit is in 1.This field is the Frame Bitrate Minimum Limit ()This field along with FrameBitrateMinUnit determines minimum allowed bits in a Frame before Multi-Pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count is less than this value. When FrameBitrateMinUnitMode is 0 (compatibility mode) bits 0:11 should be used, bits 12 and 13 should be 0.
11 [ExistsIf]Encode Only	31	**Slice Stats Streamout Enable**
	30:16	**FrameBitRateMaxDelta** Format: U15 This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28, 29 and 30 should be 0. **Value** / **Name** / **Description** 0-1024KB / / The Programmable range 0-1024KB when FrameBitRateMaxUnit is 0. 0-16380KB / / The Programmable range is 0-16380KB when FrameBitRateMaxUnit is 1. 0h / **[Default]** /
	15	**Reserved** Format: MBZ
	14:0	**FrameBitRateMinDelta** Range: The programmable range 0-1024KB When FrameBitrateMinUnit is in 32Bytes.Programmable range is 0-16380KB when FrameBitrateMinUnit is in 4Kbytes. This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. When FrameBitrateMinUnitMode is 0(compatibility mode) bits 0:11 should be used, bits 12, 13 and 14 should be 0.Note: HW requires the following condition FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior.
12	31:21	**Reserved** Format: MBZ

MFX_AVC_IMG_STATE

20 — VMD Error Logic

Value	Name	Description
0	Disable **[Default]**	
1	Enable	Error Handling

19 — Reserved

Format:	MBZ

18 — VAD Error Logic

Value	Name	Description
0	Enable **[Default]**	Error reporting ON in case of premature Slice done
1	Disable	CABAC Engine will auto decode the bitstream in case of premature slice done.

17 — Reserved

16 — Reserved

15:0 — Reserved

Format:	MBZ

13 — 31:30 Reserved

Format:	MBZ

29 — Current Picture Has Performed MMCO5

Set to 1 if the current Pic has performed the memory_management_control_operation = = 5.

28:24 — Number of Reference Frames

Format:	U5

Range: Range 0 to MaxDpbSize (=16 for Level 4.1)
Specifies the maximum number of reference frames (frames, field pairs, unpaired field) existed in the current DBP for decoding the current picture.

23:22 — Reserved

Format:	MBZ

21:16 — Number of Active Reference Pictures from L1

Format:	U6-1

Specifies the initial maximum reference index value minus 1 to access the L1 Reference List. It is extracted from PPS. It corresponds to the number of active reference pictures from L1 to decode the current picture. It can be modified by the slice header if num_ref_idx_active_override_flag is set. Only valid for B picture.

Value	Name
[0,31]	

MFX_AVC_IMG_STATE

	15:14	**Reserved** Format: MBZ
	13:8	**Number of Active Reference Pictures from L0** Format: U6-1 Specifies the initial maximum reference index value minus 1 to access the L0 Reference List. It is extracted from PPS. It corresponds to the number of active reference pictures from L0 to decode the current picture. It can be modified by the slice header if num_ref_idx_active_override_flag is set. Valid for both P and B pictures. Value: [0,31] — Name:
	7:0	**Initial QP Value** Format: S7 Range: [-26,25] Initial QP value for a Slice, extracted from PPS. It may further get modified by slice_qp_delta in slice header and mb_qp_delta in MB header.
14 [ExistsIf] Short Format only	31:24	**Log2_max_pic_order_cnt_lsb_minus4** Exists If: //Short Format Only It is a SPS syntax element, used to determine how many bits in the bitstream are used to represent pic_order_cnt_lsb syntax element in the slice header.Unsigned
	23:16	**Log2_max_frame_num_minus4** Exists If: //Short Format Only It is a SPS syntax element, used to determine how many bits in the bitstream are used to represent frame_num syntax element in the slice header.Unsigned.
	15	**deblocking_filter_control_present_flag** Exists If: //Short Format Only It is a PPS syntax element, indicates if more deblocking filter control syntax elements are present in the slice header.
	14:12	**num_slice_groups_minus1** Exists If: //Short Format Only BitField It is a PPS syntax element.Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.Desc

MFX_AVC_IMG_STATE

	11	**redundant_pic_cnt_present_flag** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only, to read-in redundant_pic_cnt, if any, but is not used by H/W, i.e. no support for redundant slice processing.
	10:8	**slice_group_map_type** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.
	7:4	**Reserved** Format: MBZ IDR flag is decoded from NAL Header Byte
	3:2	**Pic_order_cnt_type** Exists If: //Short Format Only It is a SPS syntax element.Use for Slice Header parsing only.
	1	**Delta_pic_order_always_zero_flag** Exists If: //Short Format Only It is a SPS syntax element.Use for Slice Header parsing only.
	0	**Pic_order_present_flag** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only.
15 [ExistsIf] Short Format only	31:16	**Curr Pic Frame Num** Exists If: //Short Format Only Format: U16 Derived from Slice Header syntax element
	15:0	**Slice Group Change Rate** Exists If: //Short Format Only Format: U16-1 It is a PPS syntax element Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.

MFX_AVC_IMG_STATE

16 [ExistsIf]: Short Format only	31	**Inter View Order Disable** Exists If: //Short Format Only It indicates how to append inter-view picture into initial sorted reference list. (due to ambiguity in the MVC Spec) Value / Name / Description: 0h / Default **[Default]** / View Order Ascending 1h / Disable / View ID Ascending
	30:22	**Reserved** Format: MBZ
	21:18	**Max View IDXL1** Exists If: //Short Format Only It is a PPS syntax element corresponding to Anchor/Non-Anchor Reference ListL1 It indicates the maximum number of inter-view picture for Reference List L1
	17:16	**Reserved** Format: MBZ
	15:12	**Max View IDXL0** Exists If: //Short Format Only Reference ListL0 It indicates the maximum number of inter-view picture for Reference List L0
	11:10	**Reserved** Format: MBZ
	9:0	**Current Frame View ID** Exists If: //Short Format Only It indicates the View ID of the current decoding frame
17	31:22	**Reserved** Format: MBZ
	21:16	**RhoDomain AverageMacroblockQP** Exists If: //[RhoDomain Rate Control] == 1 Format: U6
	15:9	**Reserved** Format: MBZ

MFX_AVC_IMG_STATE

	8	**Extended RhoDomain Statistics Enable** This parameter enables PAK to generate RhoDomain statistics from marcoblock QP and fractional QP computation. **Value** / **Name** / **Description** 0 / Disable / RhoDomain statistics generated from Frame Qp and no fractional QP computation. 1 / Enable / Enable MB QP based RhoDomain statistics and fractional QP computation. **Programming Notes** Reserved: MBZ
	7:6	**Reserved** Format: MBZ
	5:3	**Fractional QP offset** Format: U3 Start position from the top of the Frame where increased Quantization parameter is added. **Programming Notes** Reserved: MBZ
	2:0	**Fractional QP input** **I**n a set of 8 rows, Qp is incremented by 1 for F_qp rows..
18	31:0	**Reserved** Format: MBZ
19	31:0	**Threshold Size in Bytes** Format: U32 When a slice exceeds this value in bytes, hardware will end current slice as soon as possible and insert a new slice boundary. Note there is no guarantee that the actual slice size will meet this value, it is a hint to the HW to end the slice as soon as possible (which could be 2-5 macroblocks in the future from this detection point).
20	31:0	**Target Slice Size in Bytes** Format: U32

MFX_AVC_REF_IDX_STATE

MFX_AVC_REF_IDX_STATE

Source: VideoCS

Length Bias: 2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD mode); it is not need in decoder IT mode.
The inline data of this command is interpreted differently for encoder as for decoder. For decoder, it is interpreted as RefIdx List L0/L1 as in AVC spec., and it matches with the DXVA2 AVC API data structure for decoder in VLD mode : RefPicList[2][32] (L0:L1, 0:31 RefPic). But for encoder, it is interpreted as a Reference Index Mapping Table for L0 and L1 reference pictures. For packing the bits at the output of PAK, the syntax elements must follow the definition of RefIdxL0/L1 list according to the AVC spec. However, the decoder pipeline was designed to use a variation of that standard definition, as such a conversion (mapping) is needed to support the hardware design.
The Reference lists are needed in processing both P and B slice in AVC codec. For P-MB, only L0 list is used; for B-MB both L0 and L1 lists are needed. For a B-MB that is coded in L1-only Prediction, only L1 list is used.

Programming Notes

DXVA2 specifies that an application will create the RefPicList L0 and L1 and pass onto the driver. The content of each entry of RefPicList L0/L1[ ] is a 7-bit picture index. This picture index is the same as that of RefFrameList[ ] content. This picture index, however, is not defined the same as the frame store ID (0 to 16, 5-bits) we have implemented in H/W. Hence, driver is required to manage a table to convert between DXVA2 picture index and intel frame store ID. As such, the final RefPicList L0/L1[ ] that the driver passes onto the H/W is not the same as that defined in the DXVA2.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h PARALLEL_VIDEO_PIPE
		Format:	OpCode
	28:27	**Pipeline**	
		Default Value:	2h MFX_AVC_REF_IDX_STATE
		Format:	OpCode
	26:24	**Command Opcode**	
		Default Value:	1h AVC
		Format:	OpCode
	23:21	**SubOpcodeA**	
		Default Value:	0h MFX_AVC_REF_IDX_STATE
		Format:	OpCode
	20:16	**SubOpcodeB**	
		Default Value:	4h MFX_AVC_REF_IDX_STATE
		Format:	OpCode

MFX_AVC_REF_IDX_STATE

	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0008h Format: =n Excludes DWords 0,1
1	31:1	**Reserved** Format: MBZ
	0	**RefPicList Select** Num_ref_idx_l1_active is resulted from the specifications in both PPS and Slice Header for the current slice. However, since the full reference list L0 and/or L1 are always sent, only present flags are specified instead. This parameter is specified for Intel interface only, not present in the DXVA. **Value** / **Name** / **Description** 0 / RefPicList 0 / The list that followed represents RefList L0 (Decoder VLD mode) or Ref Idx Mapping Table L0 (Encoder PAK mode) 1 / RefPicList1 / The list that followed represents RefList L1 (Decoder VLD mode) or Ref Idx Mapping Table L1 (Encoder PAK mode)
2..9	31:0	**Reference List Entry** This set of fields is always present whenever this command is issued. It always specifies the full 32 reference pictures in the selected list, regardless they are "existing picture" or not. If a picture is non-existing, the corresponding entry should be set to all ones. Each list entry is 1 byte. A 32-bit DW can hold 4 list entries in the following format • 31:24 entry X+3 (e.g. listY_3) • 23:16 entry X+2 (e.g. listY_2) • 15:8 entry X+1 (e.g. listY_1) • 7:0 entry X (e.g. listY_0) X is replaced by the paddr[2:0] * 4 ; paddr[5:0] with 0x20 and 0x27, and Y is replaced by 0 or 1. The byte definition for a reference picture : • Bit 7 : Non-Existing - indicates that frame store index that should have been at this entry did not exist and was replaced by an index 0 (a valid entry) for error concealment • Bit 6 : Long term bit - set this reference picture to be used as long term reference • Bit 5 : Field picture flag - indicates frame/field • Bit 4:0 : Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) This is the final Reference List L0 or L1 after any reordering specified in the Slice Header as well as modified by the driver, and its indices values are all translated to the intel specification. If the reference picture is a frame (Bit5 = 1), frame store ID is always an even number.

MFX_AVC_REF_IDX_STATE		
		This list is used in outputting MV information by the BSD unit in VLD mode. DMV access also reads and writes Mvlist0 using this frame store ID. If this set of fields is interpreted as Reference Index Mapping Table L0/L1, the same field alignment is followed, i.e. 4 mapping entries per DW. Each mapping entry is one byte in size, but only the least significant 5 bits [4:0] is relevant. Driver should zero all the upper bits [7:5] for each entry.

MFX_AVC_SLICE_STATE

MFX_AVC_SLICE_STATE

Source: VideoCS

Length Bias: 2

Description

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

Programming Notes

MFX_AVC_SLICE_STATE command is not issued for AVC DXVA2 Short Format Bitstream decode, instead MFD_AVC_SLICEADDR command is executed to retrieve the next slice MB Start Address X and Y by H/W itself.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_SLICE_STATE Format: OpCode
	26:24	**Command Opcode** Default Value: 1h AVC Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h MFX_AVC_SLICE_STATE Format: OpCode
	20:16	**Command SubOpcodeB** Default Value: 3h MFX_AVC_SLICE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 8h DWORD_COUNT_n Format: =n Excludes DWords 0,1
1	31:4	**Reserved** Format: MBZ

MFX_AVC_SLICE_STATE

	3:0	**Slice Type** It is set to the value of the syntax element read from the Slice Header. **Value** / **Name** 0000b – P Slice 0001b – B Slice 0010b – I Slice 0011b-1111b – Reserved **Programming Notes** Bits[3:2] must be 0
2	31:30	**Reserved** Format: MBZ
	29:24	**Number of Reference Pictures in Inter-prediction List 1** Format: U6 This field is valid only for encoding a B Slice, for which it is expected to have at least one entry in the reference list L1; otherwise (if Slice Type is not a B Slice), this field must be set to 0. This field can be derived for a B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L1 = NumRefIdxActiveMinus1[1] + 1. **Value** / **Name** 0-32
	23:22	**Reserved** Format: MBZ
	21:16	**Number of Reference Pictures in Inter-prediction List 0** Format: U6 This field is valid for encoding a P or B Slice, for which it is expected to have at least one entry in the reference list L0; otherwise (if Slice Type is not a P or B Slice), this field must be set to 0. This field can be derived for a P or B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L0 = NumRefIdxActiveMinus1[0] + 1. **Value** / **Name** 0-32
	15:11	**Reserved** Format: MBZ
	10:8	**Log 2 Weight Denom Chroma** Format: U3 **Value** / **Name** 0-7

MFX_AVC_SLICE_STATE

	7:3	**Reserved** Format: MBZ
	2:0	**Log 2 Weight Denom Luma** Format: U3 **Description**: It is the base 2 logarithm of the denominator for all Luma weighting factors. It is set to the value of the syntax element read from the Slice Header Pred_Weight_Table(). Value: 0-7; Name:
3	31:30	**Weighted Prediction Indicator** This field indicates the Weighted Prediction mode for a P or B Slice. It is a combined field corresponding to the syntax element WeightedBiPredIdc or WeightedPredFlag read from the current active PPS. • If it is a B-Slice, these bits are interpreted as: 00b - Specifies the default weighted inter-prediction to be applied 01b - Specifies the explicit weighted inter-prediction to be applied 10b - Specifies the implicit weighted inter-prediction to be applied 11b - Reserved (not allowed) • If it is a P Slice, these bits are interpreted as: 00b - Disables weighted inter-prediction (Default weighted) 01b - Enables weighted inter-prediction (Explicit weighted) 10b - 11b - Reserved **Programming Notes**: Only when in B Slice with Weighted_Pred_Idc = 1 (explicit weighted prediction), will there be a L1 and/or a L0 weight+offset tables being sent to the BSD unit through the Slice_State command. Only when in P Slice with Weighted_Pred_Idc = 1, will there be a L0 weight+offset table being sent to the BSD. If Weighted_Pred_Idc != 1 for B Slice or Weighted_Pred_Idc =0 for P Slice, no Slice_State command should be issued to send these tables. If still being issued, the data is read but ignored. DXVA specifies Weighted_Bipred and Weighted_Pred in frame-level state. However, these two flags are combined and specified in slice level for both P and B slice type.

MFX_AVC_SLICE_STATE

	29	**Direct Prediction Type** Type of direct prediction used for B Slices. This field is valid only for Slice_Type = B Slice; otherwise, it must be set to 0. **Value** / **Name** 0 / Temporal 1 / Spatial
	28:27	**Disable Deblocking Filter Indicator** **Value** / **Name** / **Description** 00b / / FilterInternalEdgesFlag is set equal to 1 01b / / Disable all deblocking operation, no deblocking parameter syntax element is read; filterInternalEdgesFlag is set equal to 0 10b / / Macroblocks in different slices are considered not available; filterInternalEdgesFlag is set equal to 1 11b / Reserved / Not defined in AVC
	26	**Reserved** Format: MBZ
	25:24	**Cabac Init Idc[1:0]** Specifies the index for determining the initialization table used in the context variable initialization process. **Value** / **Name** 0-2 / **Programming Notes** Cabac initialization is also dependent on the field/frame picture type, Slice type, and the current SliceQP value.
	23:22	**Reserved** Format: MBZ
	21:16	**Slice Quantization Parameter** Quantization Parameter for current slice. Derived from PPS and slice_delta_qp syntax element in Slice Header. It is needed for CABAC context initialization and deblocking filter control. And it is also used as the starting QP value in the very first MB of a slice. It is in the range of unsigned integer 0 to 51, for 8-bit pixel bit-depth.
	15:12	**Reserved** Format: MBZ

MFX_AVC_SLICE_STATE

	11:8	**Slice Beta Offset Div2** Format: S3 2's Complement Range: [-6, 6] Inclusive Specifies the offset used in accessing the deblocking filter strength tables.
	7:4	**Reserved** Format: MBZ
	3:0	**Slice Alpha C0 Offset Div2** Format: S3 2's Complement Range: [-6, 6] Inclusive Specifies the offset used in accessing the deblocking filter strength tables.
4	31:24	**Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. The fields (Slice_MB_Start_Hor_Pos, Slice_MB_Start_Vert_Pos) are valid in VLD (decoding) mode only. They are ignored by hardware in decoding IT mode and encoding mode (whereas the position is provided by the per-macroblock object command). Derived **Programming Notes** Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	23:16	**Slice Horizontal Position** This field specifies the position in x-direction of the first macroblock in the Slice in unit of macroblocks. Derived **Programming Notes** Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	15	**Reserved** Format: MBZ
	14:0	**Slice Start Mb Num** Exists If: //Decoder Only The MB number (linear MB address in a picture) at the start of a Slice, it must match with the Slice Horizontal Position (Slice_MB_Start_Hor_Pos) and Vertical Position (Slice_MB_Start_Vert_Pos) in the picture. **Programming Notes** In creating the Phantom Slice for error concealment, this field should set to the total number of MB in the current picture + 1.

MFX_AVC_SLICE_STATE

5	31:24	**Reserved** Format: MBZ
	23:16	**Next Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering).
	15:8	**Reserved** Format: MBZ
	7:0	**Next Slice Horizontal Position** This field specifies the position in x-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to 0.
6 Encoder Only	31	**Rate Control Counter Enable** To enable the accumulation of bit allocation for rate control This field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields.

Value	Name
0	Disable
1	Enable

	30	**ResetRateControlCounter** To reset the bit allocation accumulation counter to 0 to restart the rate control.

Value	Name
0	Not Reset
1	Reset

	29:28	**RC Triggle Mode**

Value	Name	Description
00b	Always Rate Control	Whereas RC becomes active if sum_act > sum_target or sum_act < sum_target
01b	Gentle Rate Control	whereas RC becomes active if sum_act > upper_midpt or sum_act < lower_midpt
10b	Loose Rate Control	whereas RC becomes active if sum_act > sum_max or sum_act < sum_min
11b	Reserved	

MFX_AVC_SLICE_STATE

27:24 **RC Stable Tolerance**

Format:	U4

This field specifies the tolerance required to deactivate RC once it has been triggered.

Value	Name
0-15	

23 **RC Panic Enable**

If this field is set to 1, RC enters panic mode when sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.

Value	Name
0	Disable
1	Enable

22 **RC Panic Type**

This field selects between two RC Panic methods

Value	Name
0	QP Panic
1	CBP Panic

Programming Notes
If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

21 **MB Type Direct Conversion Disable**

Exists If:	//B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

Value	Name
0	Enable direct mode conversion
1	Disable direct mode conversion

Programming Notes
This field is zero for all other slices other than B-Slice.

MFX_AVC_SLICE_STATE

20 **MB Type Skip Conversion Disable**

Exists If:	//P-Slice or B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

Value	Name
0	Enable skip type conversion
1	Disable skip type conversion

Programming Notes
This field is zero for all other slices other than P_Slice or B-Slice. \

19 **Is Last Slice**

Description
It is used by the zero filling in the Minimum Frame Size test.
Set this only for the last slice group

Value	Name	Description
1		Current slice is the last slice of a picture
0		Current slice is NOT the last slice of a picture

18 **Reserved**

17 **Header Insertion Present in Bitstream**

Value	Name	Description
0		No header insertion into the output bitstream buffer, in front of the current slice encoded bits.
1		Header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.

Programming Notes
This need to be set only for super slice0.

16 **SliceData Insertion Present in Bitstream**

Value	Name	Description
0		No Slice Data insertion into the output bitstream buffer
1		Slice Data insertion into the output bitstream buffer is present.

Programming Notes
This bit should be set for all super-slices.

MFX_AVC_SLICE_STATE

	15	**Tail Insertion Present in bitstream** Value / Name / Description: 0 / / No tail insertion into the output bitstream buffer, after the current slice encoded bits 1 / / Tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits. **Programming Notes** This bit should only be set for the last super slice.
	14	**Reserved** Format: MBZ
	13	**EmulationByteSliceInsertEnable** To have PAK outputting SODB or EBSP to the output bitstream buffer Value / Name / Description: 0 / / outputting RBSP 1 / / outputting EBSP
	12	**CabacZeroWordInsertionEnable** To pad the end of a SliceLayer RBSP to meet the encoded size requirement. Value / Name / Description: 0 / / No Cabac_Zero_Word Insertion 1 / / Allow internal Cabac_Zero_Word generation and append to the end of RBSP (effectively can be used as an indicator for last slice of a picture, if the assumption is only the last slice of a picture needs to insert CABAC_ZERO_WORDs.
	11:8	**Reserved** Format: MBZ
	7:4	**Slice ID [3:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
	3:2	**Reserved** Format: MBZ
	1:0	**Stream ID [1:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
7 Encoder	31:29	**Reserved** Format: MBZ

MFX_AVC_SLICE_STATE

Only	28:0	**Indirect PAK-BSE Data Start Address (Write)** Exists If: //AVC Encode Mode This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes. For Write, there is no need to have a data length field. It is assumed the global memory bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access. Value: 0 - 512MB; Name:
8 Encoder Only	31:24	**Magnitude of QP Max Negative Modifier** Format: U8 This field specifies the lower limit of the QP modifier. Value: 0-51; Name:
	23:16	**Magnitude of QP Max Positive Modifier** Format: U8 This field specifies the upper limit of the QP modifier. Value: 0 - 15; Name:
	15:12	**Shrink Param - Shrink Resistance** Format: U4 This field specifies the additional points added each time decreased correction is invoked. Value: 0 - 15; Name:
	11:8	**Shrink Param - Shrink Init** Format: U4 This field specifies the initial points required to trip decreased control. Value: 0 - 15; Name:
	7:4	**Grow Param - Grow Resistance** Format: U4 This field specifies the additional points added each time increased correction is invoked. Value: 0 - 15; Name:

MFX_AVC_SLICE_STATE

	3:0	**Grow Param - Grow Init** Format: U4 This field specifies the initial points required to trip increased control. **Value** / **Name** 0 - 15
9 Encoder Only	31	**RoundInterEnable** Format: Enable When this bit is not set, RoundInter defaults to 2 to match SNB.
	30:28	**RoundInter** Format: U3 Rounding precision for Inter quantized coefficients **Value** / **Name** 000b: +1/16 **[Default]** 001b: +2/16 010b: +3/16 011b: +4/16 100b: +5/16 101b: +6/16 110b: +7/16 111b: +8/16
	27	**RoundIntraEnable** Format: Enable When this bit is not set, RoundIntra defaults to 4 to match SNB.
	26:24	**RoundIntra** Format: U3 Rounding precision for Intra quantized coefficients **Value** / **Name** 000b: +1/16 **[Default]** 001b: +2/16 010b: +3/16 011b: +4/16 100b: +5/16 101b: +6/16 110b: +7/16 111b: +8/16

MFX_AVC_SLICE_STATE

	23:20	**Correct 6** Format: U4 This field specifies the points used in the lowermost RC region when sum_act <= sum_min. Value: 0 - 15 — Name:
	19:16	**Correct 5** Format: U4 This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt. Value: 0 - 15 — Name:
	15:12	**Correct 4** Format: U4 This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target. Value: 0 - 15 — Name:
	11:8	**Correct 3** Format: U4 This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt. Value: 0 - 15 — Name:
	7:4	**Correct 2** Format: U4 This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max. Value: 0 - 15 — Name:
	3:0	**Correct 1** Format: U4 This field specifies the points used in the topmost RC region when sum_act > sum_max. Value: 0 - 15 — Name:
10 Encoder Only	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**

MFX_AVC_SLICE_STATE

	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0**

Format:	U4

If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2^{CV0}-1, they are replaced with 2^{CV0}-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped).

For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:

none	CV7	CV5	CV4
CV7	CV6	CV4	CV3
CV5	CV4	CV2	CV1
CV4	CV3	CV1	CV0

For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV7	CV6	CV5	CV4	CV3	CV3
none	CV7	CV6	CV5	CV4	CV3	CV3	CV2
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV2
CV6	CV5	CV4	CV3	CV3	CV2	CV2	CV1
CV5	CV4	CV3	CV3	CV2	CV2	CV1	CV1
CV4	CV3	CV3	CV2	CV2	CV1	CV1	CV0
CV3	CV3	CV2	CV2	CV1	CV1	CV0	CV0
CV3	CV2	CV2	CV1	CV1	CV0	CV0	CV0

For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:

none	CV6	CV3	CV1
CV7	CV6	CV3	CV1
CV5	CV4	CV2	CV0
CV5	CV4	CV2	CV0

For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV6	CV5	CV4	CV3	CV2	CV1
none	CV7	CV6	CV5	CV4	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV2	CV2	CV1
CV6	CV5	CV4	CV4	CV3	CV2	CV1	CV0
CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0

MFX_AVC_SLICE_STATE

		CV5 CV5 CV4 CV3 CV2 CV1 CV1 CV0 CV5 CV5 CV4 CV3 CV2 CV1 CV1 CV0 Value: 0 - 15 Name:

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_WEIGHTOFFSET_STATE

Source: VideoCS

Length Bias: 2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes). However, since for AVC decoder VLD and IT modes, and AVC encoder mode, the implicit weights are computed in hardware, this command is not issued. For encoder, regardless of the type of weight calculation is active for the current slice (default, implicit or explicit), they are all sent to the PAK as if they were all in explicit mode. However, for implicit weight and offset, each entry contains only a 16-bit weight and no offset (offset = 0 always in implicit mode and can be hard-coded inside the hardware).The weights (and offsets) are needed in processing both P and B slice in AVC codec. For P-MB, at most only L0 list is used; for B-MB both L0 and L1 lists may be needed. For a B-MB that is coded in L1-only Prediction, only L1 list is sent.The content of this command matches with the DXVA2 AVC API data structure for explicit prediction mode only : Weights[2][32][3][2] (L0:L1, 0:31 RefPic, Y:Cb:Cr, W:0)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_ AVC_ WEIGHTOFFSET_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 5h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 60h Excludes DWord (0,1) Format: =n Total Length - 2

MFX_AVC_WEIGHTOFFSET_STATE

1	31:1	**Reserved** Format: MBZ
	0	**Weight and Offset Select** It must be set in consistent with the WeightedPredFlag and WeightedBiPredIdc in the Img_State command. This parameter is specified for Intel interface only, not present in the DXVA. For implicit even though only one entry may be used, still loading the whole 32-entry table.

Value	Name	Description
0	Weight and Offset L0 table	The list that followed is associated with the weight and offset for RefPicList L0
1	Weight and Offset L1 table	The list that followed is associated with the weight and offset for RefPicList L1

2..97	31:0	**WeightOffset** WeightOffset[L=L0=0 or L1=1][i=0 to 31][Y=0/Cb=1/Cr=2][weight=0/offset=1] WeightOffset[L][i=0][Y=0][Weight=0], WeightOffset[L][i=0][Y=0][Offset=1] WeightOffset[L][i=0][Cb=1][Weight=0], WeightOffset[L][i=0][Cb=1][Offset=1] WeightOffset[L][i=0][Cr=2][Weight=0], WeightOffset[L][i=0][Cr=2][Offset=1]: WeightOffset[L][i=31][Y=0][Weight=0], WeightOffset[L][i=31][Y=0][Offset=1] WeightOffset[L][i=31][Cb=1][Weight=0], WeightOffset[L][i=31][Cb=1][Offset=1] WeightOffset[L][i=31][Cr=2][Weight=0], WeightOffset[L][i=31][Cr=2][Offset=1] Format for explicit: Both Weight and Offset are S15 in two's compliment, with a valid range from -128 to 128 Format for implicit: S15 This set of fields is always present whenever this command is issued. The full table, one entry for each reference picture, is always specified. Any reference list L0/L1[i] that does not exist, the corresponding weight and offset are set to 0. Weight and Offset are 2 byte each. Apair of Weight and Offset forms a dword, with Weight in the LOWER word and Offset in the HIGHER word. WeightOffset[L0=0][i=0 to 31][Y=0] (i.e. luma_weight_l0[i]) are specified for the weighting and offset factors applied to the luma prediction value for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When luma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of luma_weight_l0[i] shall be in the range of -128 to 127. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to 2luma_log2_weight_denom for RefPicList0[i]. luma_log2_weight_denom is a Slice Header syntax element. WeightOffset[L0=0][i=0 to 31][Cb=1] (i.e. chromaCb_weight_l0[i]) are specified for the weighting and offset factors applied to the chroma Cb prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of chromaCb_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCb_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i]. chroma_log2_weight_denom is a Slice Header syntax element. WeightOffset[L0=0][i=0 to 31][Cr=2] (i.e. chromaCr_weight_l0[i]) are specified for the

MFX_AVC_WEIGHTOFFSET_STATE		
		weighting and offset factors applied to the chroma Cr prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of chromaCr_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCr_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i].

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

Source: VideoCS

Length Bias: 2

This frame-level state command is used to specify all the buffer base addresses needed for the operation of the AVC Bit Stream Processing Units (for decoder, it is BSD Unit; for encoder, it is BSE Unit)

For both encoder and decoder, currently it is assumed that all codec standards can share the same BSP_BUF_BASE_STATE. The simplicity of this command is the result of moving all the direct MV related processing into the ENC Subsystem. Since all implicit weight calculations and directMV calculations are done in ENC and all picture buffer management are done in the Host, there is no need to provide POC (POC List - FieldOrderCntList, CurrPic POC - CurrFieldOrderCnt) information to PAK. For decoder, all the direct mode information are sent in a separate slice-level command (AVC_DIRECTMODE_STATE command).

In addition, in Encoder, the row stores for CABAC encoding and MB Parameters Construction (MPC) are combined into one single row store.

The row stores specified in this command do not combine with those specified in the MFC_PIPE_BUF_ADDR_STATE command for hardware simplification reason.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Pipeline Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 4h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n Total Length - 2

MFX_BSP_BUF_BASE_ADDR_STATE

1	31:6	**BSD/MPC Row Store Scratch Buffer Base Address - Read/Write** This field provides the base address of the scratch buffer used by BSD (decoder) and MPC (encoder) unit to store MB information of the previous row for coding each macroblock in the current row. It is a private buffer used by the BSD (decoder) and MPC (encoder) hardware only. Its content is not accessible by software. ThisRow Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address this Row Store. For AVC BSD, 2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF. So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes = 32,768 bytes are required. Cacheline alignment should be followed. For AVC MPC, 1 cachline for non-MBAFF, 2 cachelines for MBAFF per MB. For VC1, the BSD row store is 512-bit (one cacheline) per MB, times the number of MBs per picture MB row. **Programming Notes** This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cacheline address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage. *(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).*
	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**BSD/MPC Row Store Scratch Buffer Base Address - Read/Write [47:32]** **Description** This field is for the upper range of BSD/MPC Row Store Scratch Buffer Base Address. This field is used for 48-bit addressing.
3	31:15	**Reserved** Format: MBZ
	14:13	**BSD/MPC Row Store Scratch Buffer - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. **Value** / **Name** / **Description** 0h / TRMODE_NONE / No tiled resource

MFX_BSP_BUF_BASE_ADDR_STATE

1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12 — BSD/MPC Row Store Scratch Buffer Cache Select

This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC.

Value	Name	Description
0		Buffer going to LLC
1		Buffer going to Internal Media Storage

11 — Reserved

Format:	MBZ

10:9 — Reserved

Format:	MBZ

8:7 — BSD/MPC Row Store Scratch Buffer - Arbitration Priority Control

Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:1 — BSD/MPC Row Store Scratch Buffer - Index to Memory Object Control State (MOCS) Tables

Format:	U6

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers.

The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 — Reserved

4 — 31:6 — MPR Row Store Scratch Buffer Base Address - Read/Write (Decoder Only)

This field provides the base address of the scratch buffer used by decoder's MPR unit to store MB information of the previous row for decoding each macroblock in the current row. It is a private buffer used by the MPR hardware only. Its content is not accessible by software.

Programming Notes
The MPR Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of each macroblock to address the MPR Row Store. Except ILDB Control Data, all other operations does not cross slice boundary. This field is specified in frame-level.2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF, So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes = 32,768 bytes are required. Cacheline

MFX_BSP_BUF_BASE_ADDR_STATE

		alignment should be followed.This field is only valid for AVC decoder mode This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be cache inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage *(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).*
	5:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MPR Row Store Scratch Buffer Base Address - Read/Write [47:32]** This field is for the upper range of MPR Row Store Scratch Buffer Base Address. This field is used for 48-bit addressing.
6	31:15	**Reserved** Format: MBZ
	14:13	**MPR Row Store Scratch Buffer - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	12	**MPR Row Store Scratch Buffer Cache Select** This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC. Value / Name / Description: 0 / / Buffer going to LLC 1 / / Buffer going to Internal Media Storage
	11	**Reserved** Format: MBZ
	10:9	**Reserved** Format: MBZ

MFX_BSP_BUF_BASE_ADDR_STATE

	8:7	**MPR Row Store Scratch Buffer - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:1	**MPR Row Store Scratch Buffer - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
7	31:6	**Bitplane Read Buffer Base Address** It must be cacheline aligned (i.e. 64 bytes address boundary), so lower bit 0 to 5 are used for controlling information.(In Cantiga, this field must be dword aligned.)Bitplane buffer is a linear buffer. In VC1 Long format, it is written by an application. In VC1 Short Format, it is written and read by H/W only.For VC1 intel Long Format : it is a read-only bufferFor VC1 DXVA2 Short Format : it is a write and a read bufferThis field is only valid for VC1 decoder mode.
	5:0	**Reserved** Format: MBZ
8	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**Bitplane Read Buffer Base Address - Read/Write [47:32]** This field is for the upper range of Bitplane Read Buffer Base Address. This field is used for 48-bit addressing.
9	31:15	**Reserved** Format: MBZ
	14:13	**Bitplane Read Buffer - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h – TRMODE_NONE – No tiled resource

MFX_BSP_BUF_BASE_ADDR_STATE

		1h	TRMODE_TILEYF	4KB tiled resources
		2h	TRMODE_TILEYS	64KB tiled resources
		3h	Reserved	
	12:11	**Reserved**		
		Format:	MBZ	
	10:9	**Reserved**		
		Format:	MBZ	
	8:7	**Bitplane Read Buffer - Arbitration Priority Control**		
		Format:	U2	
		This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.		
		Value	**Name**	
		00b	Highest priority	
		01b	Second highest priority	
		10b	Third highest priority	
		11b	Lowest priority	
	6:1	**Bitplane Read Buffer - Index to Memory Object Control State (MOCS) Tables**		
		Format:	U6	
		The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.		
	0	**Reserved**		

MFX_DBK_OBJECT

MFX_DBK_OBJECT

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_DBK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h Common Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0Bh Excludes DWord (0,1) Format: =n Note: Regardless of the mode, inline data must be present in this command
1	31:6	**Pre Deblocking Source Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit).
	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ

MFX_DBK_OBJECT

	15:0	**Pre Deblocking Source Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Pre-Deblocking Source Address. This field is used for 48-bit addressing.
3	31:15	**Reserved** Format: MBZ
	14:13	**Pre Deblocking Source - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode. 0h TRMODE_NONE — No tiled resource 1h TRMODE_TILEYF — 4KB tiled resources 2h TRMODE_TILEYS — 64KB tiled resources 3h Reserved
	12:11	**Reserved** Format: MBZ
	10	**Pre Deblocking Source - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details. 1 Vertical Compression Mode
	9	**Pre Deblocking Source - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. 0 Compression Disable 1 Compression Enable
	8:7	**Pre Deblocking Source - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. 00b Highest priority 01b Second highest priority 10b Third highest priority 11b Lowest priority

MFX_DBK_OBJECT

	6:1	**Pre Deblocking Source - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
4	31:6	**Deblocking Control Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address as input MB-level deblocking parameters to control the way hardware deblock the each micro-block. One 512-bit cacheline is allocated for each Macroblock in raster scan order.
	5:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:0	**Deblocking Control Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Control Address (DeblockCntrlAddr). This field is used for 48-bit addressing.
6	31:15	**Reserved** Format: MBZ
	14:13	**Deblocking Control - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Deblocking Control - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

MFX_DBK_OBJECT

	9	**Deblocking Control - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. Value: 0 — Name: Compression Disable
	8:7	**Deblocking Control - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority
	6:1	**Deblocking Source - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
7	31:6	**Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit)
	5:0	**Reserved** Format: MBZ
8	31:16	**Reserved** Format: MBZ
	15:0	**Deblocking Destination Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Destination Address. This field is used for 48-bit addressing.
9	31:15	**Reserved** Format: MBZ
	14:13	**Deblocking Destination - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode. 0h — TRMODE_NONE — No tiled resource 1h — TRMODE_TILEYF — 4KB tiled resources

MFX_DBK_OBJECT

		2h: TRMODE_TILEYS — 64KB tiled resources 3h: Reserved
	12:11	**Reserved** Format: MBZ
	10	**Deblocking Destination - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details. Value 0: Horizontal Compression Mode
	9	**Deblocking Destination - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. Value 0: Compression Disable
	8:7	**Deblocking Destination - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:1	**Deblocking Destination - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
10	31:6	**Deblock Row Store Address** Format: GraphicsAddress[31:6] This field provides the base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of the current macroblock to address the Deblocking Filter Row Store.
	5:0	**Reserved** Format: MBZ

MFX_DBK_OBJECT

11 — 31:16 — Reserved

Format:	MBZ

Reserved for 64-bit address extension.

15:0 — Deblock Row Store Address High

Format:	GraphicsAddress[47:32]

This field is for the upper range of Deblock Row Store Address (DeblockRowStoreAddr). This field is used for 48-bit addressing.

12 — 31:15 — Reserved

Format:	MBZ

14:13 — Deblock Row Store - Tiled Resource Mode

For Media Surfaces:

This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12 — Reserved

Format:	MBZ

11 — Reserved

Format:	MBZ

10 — Deblock Row Store - Memory Compression Mode

Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode

9 — Deblock Row Store - Memory Compression Enable

Format:	Enable

Memory compression will be attempted for this surface.

Value	Name
0	Compression Disable

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td></td><td>8:7</td><td>Deblock Row Store - Arbitration Priority Control
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table></td></tr>
<tr><td></td><td>6:1</td><td>CoeffProbability StreamIn Address - Index to Memory Object Control State (MOCS) Tables
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td>Reserved</td></tr>
</table>

MFX_FQM_STATE

MFX_FQM_STATE

Source: VideoCS

Length Bias: 2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices.This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B).In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Reserved** Format: MBZ

MFX_FQM_STATE

	1:0	**AVC** Exists If: //AVC- Decoder Only **For AVC QM Type**: This field specifies which Quantizer Matrix is loaded. Value / Name: 0 – AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 1 – AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 2 – AVC_8x8_Intra_MATRIX 3 – AVC_8x8_Inter_MATRIX
	1:0	**MPEG2** Exists If: //MPEG2- Decoder Only **For MPEG2 QM Type**: This field specifies which Quantizer Matrix is loaded. Value / Name: 0 – MPEG_INTRA_QUANTIZER_MATRIX 1 – MPEG_NON_INTRA_QUANTIZER_MATRIX 2-3 – Reserved
	1:0	**JPEG** Exists If: //JPEG- Encoder Only **For JPEG QM Type**:This field specifies which Quantizer Matrix is loaded. Value / Name: 0 – JPEG_Luma_Y_QUANTIZER_MATRIX (or R) 1 – JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G) 2 – JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B) **Programming Notes** For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j]. In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.
2..33	31:0	**Forward Quantizer Matrix** Format: U32 The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

Source: VideoCS

Length Bias: 2

This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers). This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculated the corresponding memory location within the frame buffer directly.

The MFX_IND_OBJ_BASE_ADDR command sets the memory base address pointers for the corresponding Indirect Object Data Start Addresses (Offsets) specified in each OBJECT commands. The characteristic of these indirect object data is their variable size (per MB or per Slice). Hence, each OBJECT command must specify the indirect object data offset from the base address to start fetching or writing object data.

While the use of base address is unconditional, the indirection can be effectively disabled by setting the base address to zero.
For decoder, there are:

- 1 read-only per-slice indirect object in the BSD_OBJECT Command, and
- 2 read-only per-MB indirect objects in the IT_OBJECT Command.

For decoder: the Video Command Streamer (VCS) will perform the memory access bound check automatically using the corresponding MFC Indirect Object Access Upper Bound specification. If any access is at or beyond the upper bound, zero value is returned. The request to memory is still being sent, but the corresponding codec's BSD unit will detect this condition and perform the zeroing return. If the Upper Bound is turned off, the beyond bound request will return whatever on the bus (invalid data).
For encoder, there are:

- 1 read-only per-MB indirect object in the PAK_OBJECT Command, and
- 1 write-only per-slice indirect object in the PAK Slice_State Command

For encoder: whenever an out of bound address accessing request is generated, VMX will detect such requests and snap the address to the corresponding [indirect object base address + indirect data start address]. VMX will return all 0s as the data to the requestor. NotationDefinitionPhysicalAddress[n:m] Corresponding bits of a physical graphics memory byte address (not mapped by a GTT) GraphicsAddress[n:m] Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

MFX_IND_OBJ_BASE_ADDR_STATE

	28:27	**Pipeline** Default Value: 2h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	26:24	**Common Opcode** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	23:21	**Sub OpcodeA** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0018h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**MFX Indirect Bitstream Object - Base Address (Decoder and Stitch Modes)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_XXX_BSD_OBJECT command for fetching (reading) the compressed Slice Data.This field is only valid in MPEG2, AVC and VC1 decoder VLD mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect Bitstream Object - Destination Address (Decoder and Stitch Modes)[47:32]** Description This field is for the upper range of MFX Indirect Bitstream Object Base Address. This field is used for 48-bit addressing.
3	31:15	**Reserved** Format: MBZ

MFX_IND_OBJ_BASE_ADDR_STATE

	14:13	**MFX Indirect Bitstream Object - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	12:11	**Reserved** Format: MBZ
	10:9	**Reserved** Format: MBZ
	8:7	**MFX Indirect Bitstream ObjectBase - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b / Highest priority 01b / Second highest priority 10b / Third highest priority 11b / Lowest priority
	6:1	**MFX Indirect Bitstream ObjectBase - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
4	31:12	**MFX Indirect Bitstream Object - Access Upper Bound (Decoder and Stitch Modes)** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_XXX_BSD_OBJECT command for the compressed Slice Data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFX Indirect Bitstream ObjectBase Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFD_XXX_BSD_OBJECT command is set to 0.This field is only valid in MPEG2, AVC, VP8, and VC1 decoder VLD mode. **Programming Notes**

MFX_IND_OBJ_BASE_ADDR_STATE

		For **VP8 Encoder**, this field is corresponding to **MFC Indirect PAK-BSE Object - Access Upper Bound in DW24, DW25**. Please program Indirect bitstream upperbound in this field the same as DW24, DW25.
	11:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect Bitstream Object UpperBound (Decoder and Stitch Modes)[47:32]** **Description** This field is for the upper range of MFX Indirect Bitstream Object UpperBound. This field is used for 48-bit addressing. **Programming Notes** For **VP8 Encoder**, this field is corresponding to **MFC Indirect PAK-BSE Object - Access Upper Bound in DW24, DW25**. Please program Indirect bitstream upperbound in this field the same as DW24, DW25.
6	31:12	**MFX Indirect MV Object - Base Address** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the encoder MFC_AVC_PAK_OBJECT command or the decoder MFD_IT_OBJECT command for fetching the per-MB MV data.This field is only valid in AVC encoder mode or in AVC decoder IT mode
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Format: MBZ
7	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect MV Object Base Address [47:32]** **Description** This field is for the upper range of MFX Indirect MV Object Base Address. This field is used for 48-bit addressing.
8	31:15	**Reserved** Format: MBZ

MFX_IND_OBJ_BASE_ADDR_STATE

	14:13	**MFX Indirect MV Object Destination - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	12:11	**Reserved** Format: MBZ
	10:9	**Reserved** Format: MBZ
	8:7	**MFX Indirect MV Object - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b / Highest priority 01b / Second highest priority 10b / Third highest priority 11b / Lowest priority
	6:1	**MFX Indirect MV Object Desitnation - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
9	31:12	**MFX Indirect MV Object Access Upper Bound** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command for the per-MB MV data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFX Indirect MV Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command is set to 0.This field is only valid in AVC encoder mode or in AVC decoder IT mode.

MFX_IND_OBJ_BASE_ADDR_STATE

	11:0	**Reserved** Format: MBZ
10	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect MV Object UpperBound [47:32]** Description This field is for the upper range of MFX Indirect MV Object Base Address. This field is used for 48-bit addressing.
11	31:12	**MFD Indirect IT-COEFF Object - Base Address (Decoder Only)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB non-scaled coefficient data (all inverse scaling and quantization are done in hardware).This field is only valid in MPEG2, AVC and VC1 decoder IT mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Format: MBZ
12	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-COEFF Object Base Address [47:32]** Description This field is for the upper range of MFX Indirect IT-COEFF Object Base Address. This field is for the upper range of MFX Indirect MV Object Base Address. This field is used for 48-bit addressing.
13	31:15	**Reserved** Format: MBZ
	14:13	**MFD Indirect IT-COEFF - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description 0h / TRMODE_NONE / No tiled resource

MFX_IND_OBJ_BASE_ADDR_STATE

1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12:11 Reserved

Format:	MBZ

10:9 Reserved

Format:	MBZ

8:7 MFD Indirect IT-COEFF Object Desitnation - Arbitration Priority Control

Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:1 MFD Indirect IT-COEFF Object Desitnation - Index to Memory Object Control State (MOCS) Tables

Format:	U6

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers.
The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 Reserved

14 31:12 MFD Indirect IT-COEFF Object - Access Upper Bound (Decoder Only)

Format:	GraphicsAddress[31:12]

This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB non-scaled coefficient data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFD Indirect IT-COEFF Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect COEFF Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in MPEG2, AVC and VC1 decoder IT mode.

11:0 Reserved

Format:	MBZ

15 31:16 Reserved

Format:	MBZ

Reserved for 64-bit address extension.

MFX_IND_OBJ_BASE_ADDR_STATE

	15:0	**MFD Indirect IT-COEFF Object UpperBound [47:32]** Description This field is for the upper range of MFX Indirect IT-COEFF Object UpperBound. This field is for the upper range of MFX Indirect MV Object Base Address. This field is used for 48-bit addressing.
16	31:12	**MFD Indirect IT-DBLK Object - Base Address (Decoder Only)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB Deblocking filter control data.This field is only valid in AVC decoder IT mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Format: MBZ
17	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-DBLK Object Base Address [47:32]** Description This field is for the upper range of MFX Indirect IT-DBLK Object Base Address. This field is used for 48-bit addressing.
18	31:15	**Reserved** Format: MBZ
	14:13	**MFD Indirect IT-DBLK Object Destination - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	12:11	**Reserved** Format: MBZ
	10:9	**Reserved** Format: MBZ

MFX_IND_OBJ_BASE_ADDR_STATE

	8:7	**MFD Indirect IT-DBLK Object - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:1	**MFD Indirect IT-DBLK Object Desitnation - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
19	31:12	**MFD Indirect IT-DBLK Object - Access Upper Bound (Decoder Only)** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB Deblocking filter control data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFD Indirect IT-DBLK Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Deblocking Control Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in AVC decoder IT mode.
	11:0	**Reserved** Format: MBZ
20	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-DBLK Object UpperBound [47:32]** Description: This field is for the upper range of MFX Indirect IT-DBLK Object UpperBound. This field is used for 48-bit addressing.

MFX_IND_OBJ_BASE_ADDR_STATE

21	31:12	**MFC Indirect PAK-BSE Object - Base Address (Encoder Only)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the write-only indirect data object pointed in the PAK_SLICE_STATE command for writing out the compressed bitstream.This field is only valid in AVC encoder mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Format: MBZ
22	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFC Indirect PAK-BSE Object Base Address [47:32]** Description This field is for the upper range of MFX Indirect PAK-BSE Object Base Address. This field is used for 48-bit addressing.
23	31:15	**Reserved** Format: MBZ
	14:13	**MFC Indirect PAK-BSE Object Destination - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.
	12:11	**Reserved** Format: MBZ
	10:9	**Reserved** Format: MBZ

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

MFX_IND_OBJ_BASE_ADDR_STATE

	8:7	**MFC Indirect PAK-BSE Object Desitnation - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:1	**MFC Indirect PAK-BSE Object Desitnation - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
24	31:12	**MFC Indirect PAK-BSE Object - Access Upper Bound (Eecoder Only)** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the PAK_SLICE_STATE command for the per-slice output bitstream. Indirect data accessed at this address and beyond will be blocked by the hardware and ignored. Setting this field to 0 will cause this range check to be ignoredIf non-zero, this address must be greater than the MFC Indirect PAK-BSE Object Base Address state.This field is only valid in AVC encoder mode. **Programming Notes** For VP8 Encoder, this field should be programmed the same at both DW4, DW5 **MFX Indirect Bitstream Object - Access Upper Bound** as well as DW24, DW25.
	11:0	**Reserved** Format: MBZ
25	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.

MFX_IND_OBJ_BASE_ADDR_STATE		
	15:0	**MFC Indirect PAK-BSE Object UpperBound [47:32]** **Description** This field is for the upper range of MFC Indirect PAK-BSE Object UpperBound This field is used for 48-bit addressing. **Programming Notes** For VP8 Encoder, this field should be programmed the same at both DW4, DW5 **MFX Indirect Bitstream Object - Access Upper Bound** as well as DW24, DW25.

MFX_JPEG_HUFF_TABLE_STATE

MFX_JPEG_HUFF_TABLE_STATE

Source: VideoCS

Length Bias: 2

This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image. A Huffman table will be sent to H/W only when it is loaded from bitstream.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 033Dh Excludes DWord (0,1) Format: =n Total Length - 2
1	31:1	**Reserved** Format: MBZ
	0	**HuffTableID (1-bit)** Identifies the huffman table. Value: 0 — Name: Y — Description: Huffman table for Y

MFX_JPEG_HUFF_TABLE_STATE		
2..4	31:0	**DC_BITS (12 8-bit array)** The number of DC Huffman codes of length i, where i is 1~12
5..7	31:0	**DC_HUFFVAL (12 8-bit array)** The value associated with each DC Huffman code of length i.
8..11	31:0	**AC_BITS (16 8-bit array)** the list of Li, number of Huffman codes of length i, where i is 1~16
12..51	31:0	**AC_HUFFVAL (160 8-bit array)** the list of Vi,j, the value associated with each Huffman code of length i
52	31:16	**Reserved** Format: MBZ
	15:0	**AC_HUFFVAL(2-8 bit array)** In AC table, BITS can have up to 16-bit codeword. Li can be 0 ~ 162. HUFFVAL will have a list of likely random distributed values

MFX_JPEG_PIC_STATE

MFX_JPEG_PIC_STATE

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 0001h / **[Default]** / Excludes DWord (0,1)
1	31	**Reserved** Exists If: //Encoder Only Format: MBZ
	30:26	**Pixels In Horizontal Last MCU** Exists If: //Encoder Only The number of pixels in the last MCU in a row MCUs. This information is used for completion of partial MCU.

MFX_JPEG_PIC_STATE

31:21 Reserved

Exists If:	//Decoder Only
Format:	MBZ

25:21 Pixels In Vertical Last MCU

Exists If:	//Encoder Only

The number of pixels in the last MCU in a column MCUs. This information is used for completion of partial MCU.

20 Vertical Up-Sampling Enable

Exists If:	//Decoder Only

Only applied to chroma blocks. This flag is used for 2:1 vertical up-sampling for chroma 420 and outputting chroma422 YUY2 or UYVY format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV420, and **OutputFormatYUV** should be set to YUY2 or UYVY.

Value	Name	Description
0b		no up-sampling
1b		2:1 vertical up-sampling

19 Reserved

Exists If:	//Decoder Only

18 Horizontal Down-Sampling Enable

Exists If:	//Decoder Only

Only applied to chroma blocks. This flag is used for 2:1 horizontal down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV422V_2Y or YUV422V_4Y, and **OutputFormatYUV** should be set to NV12.

Value	Name	Description
0b		no down-sampling
1b		2:1 horizonatl down-sampling

17 Vertical Down-Sampling Enable

Exists If:	//Decoder Only

Only applied to chroma blocks. This flag is used for 2:1 vertical down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV422H_2Y or YUV422H_4Y, and **OutputFormatYUV** should be set to NV12.

Value	Name	Description
0b		no down-sampling
1b		2:1 vertical down-sampling

MFX_JPEG_PIC_STATE

	16	**Average Down Sampling** Exists If: //Decoder Only This flag is used to select a down-sampling method when **VertDownSamplingEnb** or **HoriDownSamplingEnb** is set to 1. **Value** / **Name** / **Description** 0b / / Drop every other line (or column) pixels 1b / / Average neighboring two pixels
	20:12	**Reserved** Exists If: //Encoder Only Format: MBZ
	15:12	**Reserved** Exists If: //Decoder Only Format: MBZ
	11:8	**Output Format YUV** Exists If: //Decoder Only This field specifies the surface format to write the decoded JPEG image.Note that any non-interleaved JPEG input should be set to "0000". For the interleaved input Scan data, it can be set either "0000" or the corresponding format. **Value** / **Name** / **Description** 0000b / / 3 separate plane for Y, U, and V respectively 0001b / / NV12 for chroma 4:2:0 0010b / / UYVY for chroma 4:2:2 0011b / / YUY2 for chroma 4:2:2 **Programming Notes** The **MFX_SURFACE_STATE** command should be set accordingly for each **OutputFormatYUV**. For NV12, **Surface Format** = 4 (PLANAR_420_8) For YUY2, **Surface Format** = 0 (YCRCB_NORMAL) For UYVY, **Surface Format** = 3 (YCRCB_SWAPY) NV12 (0001b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases • **InputFormatYUV** is YUV420 and **VertDownSamplingEnb** is disalbed • **InputFormatYUV**is YUV422H_2Y or YUV422H_4Y, and **VertDownSamplingEnb** is enabled UYVY (0010b) and YUY2 (0011b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases • **InputFormatYUV** is YUV420 and **VertUpSamplingEnb** is enabled • **InputFormatYUV** is YUV422H_2Y or YUV422H_4Y and **VertUpSamplingEnb** is disabled

MFX_JPEG_PIC_STATE

11:8 **Input Surface Format YUV**

Exists If:	//Encoder Only

This field specifies the surface format to read a YUV image data

Value	Name	Description
0000b		Reserved
0001b	NV12	NV12 for chroma 4:2:0
0010b	UYVY	UYVY for chroma 4:2:2
0011b	YUY2	YUY2 for chroma 4:2:2
0100b	Y8	Y8 for chroma400 Y-only image
0101b	RGB	RGB or YUV for chroma 4:4:4

Programming Notes
This field should be set accordingly for **SurfaceFormat** in MFX_SURFACE_STATE command.
R8G8B8A8_UNORM in this field is used for encoding RGB and YUV chroma 4:4:4. For RGB input, any order of image components R, G, B (e.g., RGB, GBR, BGR, YUV) will be acceptable as far as the order of Quantization tables and Huffman tables match the order of image components.

7:6 **Reserved**

Exists If:	//Decoder Only
Format:	MBZ

7:6 **Reserved**

Exists If:	//Encoder Only
Format:	MBZ

5:4 **Rotation**

Exists If:	//Decoder Only

Value	Name	Description
00b		no rotation
01b		rotate clockwise 90 degree
10b		rotate counter-clockwise 90 degree (same as rotating 270 degree clockwise)
11b		rotate 180 degree (NOT the same as flipped on the x-axis)

Programming Notes
Rotation can be set to 01b, 10b, or 11b when OutputFormatYUV is set to 0000b. For other OutputFormatYUV, Rotation is not allowed.

MFX_JPEG_PIC_STATE

	5:3	**Reserved** Exists If: //Encoder Only Format: MBZ
	3	**Reserved** Exists If: //Decoder Only Format: MBZ
	2:0	**Input Format YUV** Exists If: //Decoder Only Format: U3

Value	Name	Description
0	**[Default]**	YUV400 (grayscale image)
1		YUV420
2		YUV422H_2Y (Horizontally chroma 2:1 subsampled) - horizontal 2 Y-block, 1U and 1V
3		YUV444
4		YUV411
5		YUV422V_2Y (Vertically chroma 2:1 subsampled) - vertical 2 Y-blocks, 1U and 1V
6		YUV422H_4Y - 2x2 Y-blocks, vertical 2U and 2V
7		YUV422V_4Y - 2x2 Y-blocks, horizontal 2U and 2V

	2:0	**Output MCU Structure** Exists If: //Encoder Only Output MCU Structure(**OutputMcuStructure**) should be set accordingly for each Input Surface Format YUV(**InputSurfaceFormatYUV**): • If **InputSurfaceFormatYUV** is set to NV12, **OutputMCUStructure** is set to YUV420. • If **InputSurfaceFormatYUV** is set to UYVY or YUY2, **OutputMCUStructure** is set to YUV422H_2Y. • If **InputSurfaceFormatYUV** is set to Y8, **OutputMCuStructure** is set to YUV400. • If **InputSurfaceFormatYUV** is set to RGB (or GBR, BGR, YUV), **OutputMCuStructure** is set to RGB. • If **InputSurfaceFormatYUV** is set to RGB, the order of encoded blocks in MCU will be same as the order of input image components. If the order of input image components is RGB (or GBR, BGR, YUV), then the order of blocks will be RGB (or GBR, BGR, YUV respectively).

Value	Name	Description
0	YUV400	Grayscale Image

MFX_JPEG_PIC_STATE

		1	YUV420	Both horizontally and vertically chroma 2:1 subsampled
		2	YUV422H_2Y	Horizontally chroma 2:1 subsampled - horizontal 2 Y-blocks, 1 U and 1 V block
		3	RGB	RGB or YUV444: No subsample
		4		
		5		
		6		
		7		

2 — 31:30 — Reserved

Exists If:	//Decoder Only
Format:	MBZ

31:29 — Reserved

Exists If:	//Encoder Only
Format:	MBZ

29 — Output Pixel Normalize

Exists If:	//Decoder Only

JPEG decoded output pixels for Y and U/V in order to adjust display YUV range.

Value	Name	Description	Exists If
0		No Normalization	
1		Normalize output pixels from [0,255] to [16,235]	//Y
1		Normalize output pixels from [0,255] to [16,239]	//U/V

28:16 — Frame Height In Blocks Minus 1

Exists If:	//Decoder Only
Format:	U13-1

(The number of blocks in height) - 1.
This value is calculated using the number of lines Y and vertical sampling factor of the first component V_1 in Frame header. See the note following this table.
For interleaved components, (((Y + (V_1*8 -1)) / (V_1*8)) * V_1) - 1, where "/" is integer division.
For non-interleaved components, ((Y + 7) / 8) - 1.

28:16 — Frame Height In Blks Minus 1

Exists If:	//Encoder Only
Format:	U13-1

(The number of blocks in height) - 1.
This value is calculated using the number of lines Y and vertical sampling factor of the first component V1 in Frame header. See the note following this table.

For interleaved components: (((Y + (V1*8 -1)) / (V1*8)) * V1) - 1
For non-interleaved components: ((Y + 7) / 8) - 1

MFX_JPEG_PIC_STATE

15:13 **Reserved**

Exists If:	//Decoder Only
Format:	MBZ

15:13 **RoundingQuant**

Exists If:	//Encoder Only

Rounding value applied to quantization output

Value	Name	Description
000b	**[Default]**	1/2
001b		(1/2 - 1/128)
010b		(1/2 + 1/128)
011b		(1/2 - 1/64)
100b		(1/2 + 1/64)
101b		(1/2 - 1/32)
110b		(1/2 - 1/16)
111b		(1/2 - 1/8)

12:0 **Frame Width In Blocks Minus 1**

Exists If:	//Decoder Only
Format:	U13-1

(The number of blocks in width) - 1.
This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H_1 in Frame header. See the note following this table.
For interleaved components, (((X + (H_1 *8 -1)) / (H_1 *8)) * H_1) - 1.
For non-interleaved components, ((X + 7) / 8) - 1.

12:0 **Frame Width In Blks Minus 1**

Exists If:	//Encoder Only
Format:	U13-1

(The number of blocks in width) - 1.
This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H1 in Frame header. See the note following this table.

For interleaved components: (((X + (H1 *8 -1)) / (H1 *8)) * H1) - 1
For non-interleaved components: ((X + 7) / 8) - 1

MFX_MPEG2_PIC_STATE

MFX_MPEG2_PIC_STATE

Source: VideoCS

Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands. For MPEG-2 the encoder is called per slice-group, however the picture state is called per picture.Notice that a slice-group is a group of consecutive slices that no non-trivial slice headers are inserted in between.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1)= 00Bh, used for normal decode and encode mode000h, a special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware. Format: =n Total Length - 2
1	31:28	**f_code[1][1].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	27:24	**f_code[1][0].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details

MFX_MPEG2_PIC_STATE

	23:20	**f_code[0][1]** Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	19:16	**f_code[0][0]** Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	15:14	**Intra DC Precision** Format: U2 See ISO/IEC 13818-2 6.3.10 for details.
	13:12	**Picture Structure** This field specifies whether the picture is encoded in the form of a frame picture or one field (top or bottom) picture. See ISO/IEC 13818-2 6.3.10 for details.Format = MPEG_PICTURE_STRUCTURE00 = Reserved01 = MPEG_TOP_FIELD10 = MPEG_BOTTOM_FIELD11 = MPEG_FRAME
	11	**TFF (Top Field First)** When two fields are stored in a picture, this bit indicates if the top field is the first field.For a frame P picture, the value 1 indicates that the top field of the reconstructed frame is the first field output by the decoding process, the same as defined in ISO/IEC 13818-2 6.3.10. Particularly, it is used by the hardware to calculate derivative motion vectors from the dual-prime motion vectors.For a field P picture, hardware uses this bit together with the Picture Structure to determine if the current picture is the Second Field. In this case, the definition of this bit differs from ISO/IEC 13818-2 6.3.10 - software must derive the value for this bit according to the following relation:Picture Structure = top fieldPicture Structure = bottom fieldSecond Field = 0TFF = 1TFF = 0Second Field = 1TFF = 0TFF = 1
	10	**Frame Prediction Frame DCT** This field provides constraints on the DCT type and prediction type. It affects the syntax of the bitstream.
	9	**Concealment Motion Vector Flag** This field indicates if the concealment motion vectors are coded in intra macroblocks. It affects the syntax of the bitstream.
	8	**Quantizer Scale Type** Format: MPEG_Q_SCALE_TYPE This field specifies the quantizer scaling type. Value / Name / Description 0h / / MPEG_QSCALE_LINEAR 1h / / D MPEG_QSCALE_NONLINEAR esc
	7	**Intra VLC Format** This field is used by VLD
	6	**Scan Order** Format: MPEG_INVERSESCAN_TYPE This field specifies the Inverse Scan method for the DCT-domain coefficients in the

MFX_MPEG2_PIC_STATE

blocks of the current picture.

Value	Name	Description
0h		MPEG_ZIGZAG_SCAN
1h		MPEG_ALTERNATE_VERTICAL_SCAN

5:0 **Reserved**

2 31 **I Slice Concealment Mode**

Exists If:	//Decoder

This field controls how MPEG decoder handles MB concealment in I Slice

Value	Name	Description
0h	Intra Concealment	Using Coefficient values to handle MB concealment
1h	Inter Concealment	Using Motion Vectors to handle MB concealment

Programming Notes
If this field is set to "1", driver must provide a valid forward reference picture (both top and bottom Field must be valid)

30 **Reserved**

Format:	MBZ

29:28 **P/B Slice Concealment Mode**

Exists If:	//Decoder

This field controls how MPEG decoder handles MB concealment in P/B Slice.

Value	Name	Description
00b	INTER	If left MB is NOT Intra MB type (including skipMB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use forward reference (same polarity for field pic) with MV final values set to 0.
01b	LEFT	If left MB is NOT Intra MB type (including skipMB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data)
10b	ZERO	Always use forward reference (same polarity for field pic) with MV final values set to 0 (Macroblock is concealed as INTER coded)
11b	INTRA	Use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data

27 **Reserved**

Format:	MBZ

MFX_MPEG2_PIC_STATE

26:25 **P/B Slice Predicted BiDir Motion Type Override - Bi-direction MV Type Override**

Exists If:	//Decoder

This field is only applicable if the Concealment Motion Type is predicted to be Bi-directional.
(It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB is a bi-directional MB).

Value	Name	Description
0h	BID	Keep Bi-direction Prediction
1h	RESERVED	
2h	FWD	Only use Forward Prediction (Backward MV is forced to invalid
3h	BWD	Only use Backward Prediction (Forward MV is forced to invalid)

24 **P/B Slice Predicted Motion Vector Override Final MV value Override**

Exists If:	//Decoder

This field is only applicable if the Concealment Motion Vectors are non-zero.
It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB has non-zero motion vectors).

Value	Name	Description
0h	Predicted	Motion Vectors use predicted values
1h	ZERO	Motion Vectors force to 0

23:15 **Reserved**

Format:	MBZ

14 **LoadSlicePointerFlag - LoadBitStreamPointerPerSlice**

Exists If:	//Encoder

To support multiple slice picture and additional header/data insertion before and after an encoded slice.When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream.When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0h		Load BitStream Pointer only once for the first slice of a frame
1h		Load/reload BitStream Pointer only once for the each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field

13 **Reserved**

Format:	MBZ

12 **Reserved**

Format:	MBZ

MFX_MPEG2_PIC_STATE

	11	**Reserved** Format: MBZ
	10:9	**Picture Coding Type** Format: MPEG_PICTURE_CODING_TYPE This field identifies whether the picture is an intra-coded picture (I), predictive-coded picture (P) or bi-directionally predictive-coded picture (B). See ISO/IEC 13818-2 6.3.9 for details. **Value** / **Name** 00b: Reserved 01b: MPEG_I_PICTURE 10b: 10 = MPEG_P_PICTURE 11b: MPEG_B_PICTURE
	8:2	**Reserved** Format: MBZ
	1	**MismatchControlDisabled** These 2 bits flag disables mismatch control of the inverse transformation for some specific cases during reference reconstruction. **Value** / **Name** / **Description** 00b: Mismatch control applies to all MBs 01b: Disable mismatch control to all intra MBs whose all AC-coefficients are zero. 10b: Disable mismatch control to all MBs whose all AC-coefficients are zero. 11b: Disable mismatch control to all MBs.
	0	**Disable Mismatch** To disable MPEG2 IDCT fixed point arithmetic correction
3	31	**Slice Concealment Disable Bit** Exists If: //Decode If VINunit detects the next slice starting position is either out-of-bound or smaller than or equal to the current slice starting position, VIN will set the current slice to be 1 MB and force VMDunit to do slice concealment on the next slice. This bit will disable this feature and the MB data from the next slice will be decoded from bitstream. **Value** / **Name** / **Description** 0h: Enable **[Default]** — VIN will force next slice to be concealment if detects slice boundary error 1h: Disable — VIN will not force next slice to be in concealment **Programming Notes**

MFX_MPEG2_PIC_STATE

		Driver has an option to detect the scenario given in description (above) and remove the second (out-of-order) slice. In this case, hardware will decode the first slice in completion and do concealment till the third slice. It should yield a picture with better quality this way.
	30:29	**Reserved** Format: MBZ
	28:24	**Reserved**
	23:16	**FrameHeightInMBsMinus1[7:0] (Picture Height in Macroblocks)** Format: U8
	15:8	**Reserved** Format: MBZ for future supporting width > 4K
	7:0	**FrameWidthInMBsMinus1[7:0] (Picture Width in Macroblocks)** Format: U8
4	31:16	**MinFrameWSize** Format: U16 - Minimum Frame Size [15:0] (16-bit) (Encoder Only)Mininum Frame Size is specified to compensate for intel Rate ControlCurrently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Mininum Frame Size, is always less than maximum frame size FrameBitRateMax (DWORD 10 bits 29:16). This field is reserved in Decode mode.

Value	Name	Description
[0,0003FFFFh]		The programmable range when MinFrameWSizeUnits is 00.
[0,000FFFFFh]		The Programmable range when MinFrameWSizeUnits is 01.
[0,03FFFFFFh]		The Programmable range when MinFrameWSizeUnits is 10.
[0, FFFFFFFFh]		The Programmable range when MinFrameWSizeUnits is 11.
0h	**[Default]**	

	15	**Reserved** Format: MBZ
	14:12	**RoundInterAC,** rounding precision for non-Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	11	**Reserved** Format: MBZ

MFX_MPEG2_PIC_STATE

	10:8	**RoundIntraAC** Format: U3 rounding precision for Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	7	**Reserved** Format: MBZ
	6:4	**RoundInterDC** rounding Precision for non-Intra-DC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	3	**Reserved** Format: MBZ
	2:1	**RoundIntraDC** rounding Precision for Intra-DC00: +1/801: +2/810: +3/811: +4/8
	0	**Reserved**
5	31:17	**Reserved** (for future Mask bits)
	16	**FrameSizeControlMask** Frame size conformance maskThis field is used when MacroblockStatEnable is set to 1.

Value	Name	Description
0h		Do not change Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control
1h		Replace Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control values in MFC_IMAGE_STATUS control register.

	15:13	**Reserved**
	12	**InterMBForceCBPZeroControlMask** Format: U1 Inter MB Force CBP ZERO mask.

Value	Name	Description
[0, FFFFFFFFh]		
0h		No effect
1h		Zero out all A/C coefficients for the inter MB violating Inter Confirmance

MFX_MPEG2_PIC_STATE

	11:10	**MinFrameWSizeUnits** This field is the Minimum Frame Size Units **Value** / **Name** / **Description** 00b / compatibility mode / Minimum Frame Size is in old mode (words, 2bytes) 01b / 16 byte / Minimum Frame Size is in 16bytes 10b / 4Kb / Minimum Frame Size is in 4Kbytes 11b / 16Kb / Minimum Frame Size is in 16Kbytes
	9	**MBRateControlMask** MB Rate Control conformance maskThis field is ignored when MacroblockStatEnable is disabled or MB level Rate control flag for the current MB is disable in Macroblock Status Buffer. **Value** / **Name** / **Description** 0h / / Do not change QP values of inter macroblock with suggested QP values in Macroblock Status Buffer 1h / / Apply RC QP delta for all macroblock
	8	**Reserved**
	7	**Reserved** Format: MBZ
	6:4	**Reserved**
	3	**FrameBitRateMinReportMask** This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin. **Value** / **Name** / **Description** 0h / Disable / Do not update bit0 of MFC_IMAGE_STATUS control register. 1h / Enable / set bit0 and bit 1of MFC_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit rate Minimum limit.
	2	**FrameBitRateMaxReportMask** This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax. **Value** / **Name** / **Description** 0h / Disable / Do not update bit0 of MFC_IMAGE_STATUS control register. 1h / Enable / set bit0 and bit 1 of MFC_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit rate Maximum limit.
	1	**InterMBMaxSizeReportMask** This is a mask bit controlling if the condition of any inter MB in the frame exceeds InterMBMaxSize. **Value** / **Name** / **Description**

MFX_MPEG2_PIC_STATE

		0h: Do not update bit0 of MFC_IMAGE_STATUS control register. 1h: set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Inter MB Conformance Max size limit.
	0	**IntraMBMaxSizeReportMask** This is a mask bit controlling if the condition of any intra MB in the frame exceeds IntraMBMaxSize. **Value / Name / Description** 0h: Do not update bit0 of MFC_IMAGE_STATUS control register. 1h: set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Intra MB Conformance Max size limit.
6 [ExistsIf]Encode Only	31:28	**Reserved** Format: MBZ
	27:16	**InterMBMaxSize** Default Value: FFFh This field, Inter MB Conformance Max size limit,indicates the allowed max bit count size for Inter MB
	15:12	**Reserved** Format: MBZ
	11:0	**IntraMBMaxSize** Default Value: FFFh This field, Intra MB Conformance Max size limit,indicates the allowed max bit count size for Intra MB
7	31:1	**Reserved** Format: MBZ
	0	**VSL top MB Trans8x8flag** Exists If: //Encode Only **Value / Name / Description** 0 / Disable: VSL will only fetch the current MB data. 1 / Enable: When this bit is set VSL will make extra fetch to memory to fetch the MB data for top MB.

MFX_MPEG2_PIC_STATE

8 [ExistsIf]Encode Only	31:24	**SliceDeltaQPMax[3]** Format: S7 This field is the Slice level delta QP for total bit-count above FrameBitRateMax - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta»3). Range: [-30,30] **Value** / **Name** 0h — Disable 1h — Enable
	23:16	**SliceDeltaQPMax[2]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/8 and below 1/ 4 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and ¼ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»3), (FrameBitRateMax+ FrameBitRateMaxDelta»2).
	15:8	**SliceDeltaQPMax[1]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above1/ 4 and below 1/2 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between ¼ and ½ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»2), (FrameBitRateMax+ FrameBitRateMaxDelta»1).
	7:0	**SliceDeltaQPMax[0]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta , i.e., in the range of

MFX_MPEG2_PIC_STATE

		((FrameBitRateMax+ FrameBitRateMaxDelta»1), infinite).
9 [ExistsIf]Encode Only	31:24	**SliceDeltaQPMin[3]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for total bit-count below FrameBitRateMin - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»3), FrameBitRateMin).
	23:16	**SliceDeltaQPMin[2]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count below FrameBitRateMin - below 1/ 8 and above 1/ 4This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»2), (FrameBitRateMin- FrameBitRateMinDelta»3)).
	15:8	**SliceDeltaQPMin[1]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count below FrameBitRateMin- below 1/4 and above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»1), (FrameBitRateMin- FrameBitRateMinDelta»2)).
	7:0	**SliceDeltaQPMin[0]** Format: S7 Range: [-30,30] This field is the Slice Level Delta QP for bit-count below FrameBitRateMin - below 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta , i.e., in the range of [0, (FrameBitRateMin- FrameBitRateMinDelta»1).

MFX_MPEG2_PIC_STATE

10 [ExistsIf]Encode Only

31 **FrameBitrateMaxUnit**

This field is the Frame Bitrate Maximum Limit Units.

Value	Name	Description
0h	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMaxUnitMode is 1 and in units of 128 Bytes if FrameBitrateMaxUnitMode is 0
1h	Kilobyte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0

30 **FrameBitrateMaxUnitMode**

BitFiel This field is the Frame Bitrate Maximum Limit Units.dDesc

Value	Name	Description
0h	Compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)
1h	New mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)

29:16 **FrameBitRateMax**

This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28 and 29 should be 0.

Value	Name	Description
0-512KB		The programmable range 0-512KB when FrameBitrateMaxUnit is 0.
0-8190KB		The programmable range 0-8190KB when FrameBitrateMaxUnit is 1.

15 **FrameBitrateMinUnit**

This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description
0h	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMinUnitMode is 1 and in units of 128 Bytes if FrameBitrateMinUnitMode is 0
1h	KiloByte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0

14 **FrameBitrateMinUnitMode**

This field is the Frame Bitrate Minimum Limit Units.ValueNameDescriptionProject

Value	Name	Description
0h	compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)
1h	New Mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)

MFX_MPEG2_PIC_STATE

	13:0	**FrameBitRateMin** This field is the Frame Bitrate Minimum Limit ()This field along with FrameBitrateMinUnit determines minimum allowed bits in a Frame before Multi-Pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count is less than this value. When FrameBitrateMinUnitMode is 0 (compatibility mode) bits 0:11 should be used, bits 12 and 13 should be 0. Range: The programmable range 0-512KB When FrameBitrateMinUnit is in 0. Programmable range is 0-8190 KB when FrameBitrateMinUnit is in 1
11 [ExistsIf]Encode Only	31	**Reserved** Format: MBZ
	30:16	**FrameBitRateMaxDelta** Default Value: 0h Access: None Format: U15 This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. The programmable range is either 0- 512KB or 4MBB in FrameBitrateMaxUnit of 128 Bytes or 16KB respectively. This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28, 29 and 30 should be 0.
	15	**Reserved** Format: MBZ
	14:0	**FrameBitRateMinDelta** This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. When FrameBitrateMinUnitMode is 0(compatibility mode) bits 0:11 should be used, bits 12, 13 and 14 should be 0.Note: HW requires the following condition FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior. **Value** / **Name** / **Description** 0-1024KB / / The programmable range 0-1024KB When FrameBitrateMinUnit is in 32Bytes. 0-16380KB / / Programmable range is 0-16380KB when FrameBitrateMinUnit is in 4Kbytes.
12	31:21	**Reserved** Format: MBZ

MFX_MPEG2_PIC_STATE

	20	**VMD Error Logic** Value / Name / Description 0 / Disable **[Default]** / 1 / Enable / Error Handling
	19	**Reserved** Format: / MBZ
	18	**VAD Error Logic** Value / Name / Description 0 / Enable **[Default]** / Error reporting ON in case of premature Slice done 1 / Disable / CABAC Engine will auto decode the bitstream in case of premature slice done.
	17	**Reserved**
	16	**RESERVED**
	15:0	**Reserved** Format: / MBZ

MFX_PAK_INSERT_OBJECT

MFX_PAK_INSERT_OBJECT

Source:	VideoCS
Length Bias:	2

Description

The MFX_PAK_INSERT_OBJECT command is the first primitive command for the AVC, MPEG2, JPEG, and VP8 Encoding Pipeline.

This command is issued to setup the control and parameters of inserting a chunk of compressed/encoded bits into the current bitstream output buffer starting at the specified bit locationto perform the actual insertion by transferring the command inline data to the output buffer max, 32 bits at a time.
It is a variable length command as the data to be inserted are presented as inline data of this command. It is a multiple of 32-bit (1 DW), as the data bus to the bitstream buffer is 32-bit wide.
Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid H.264 bitstream.
Internally, MFX hardware will keep track of the very last two bytes' (the very last byte can be a partial byte) values of the previous insertion. It is required that the next Insertion Object Command or the next PAK Object Command to perform the start code emulation sequence check and prevention 0x03 byte insertion with this end condition of the previous insertion.
Hardware will keep track of an output bitstream buffer current byte position and the associated next bit insertion position index. Data to be inserted can be a valid H.264 NAL units or a partial NAL unit. Certain NAL unit has a minimum byte size requirement. As such the hardware will optionally (enabled by STATE Command) determines the number of CABAC_ZERO_WORD to be inserted to the end of the current NAL, based on the minimum byte size of a NAL and the actual bin count of the encoded Slice. Since prior to the CABAC_ZERO_WORD insertion, the RBSP or EBSP is already byte-aligned, so each CABAC_ZERO_WORD insertion is actually a 3-byte sequence 0x00 00 03. The inline data may have already been processed for start code emulation byte insertion, except the possibility of the last 2 bytes plus the very last partial byte (if any). Hence, when hardware performing the concatenation of multiple consecutive insertion commands, or concatenation of an insertion command and a PAK object command, it must check and perform the necessary start code emulation byte insert at the junction.The inline data is required to be byte aligned on the left (first transmitted bit order) and may or may not be byte aligned on the right (last transmitted bits).
The command will specify the bit offset of the last valid DW.Each insertion state command defines a chunk of bits (compressed data) to be inserted at a specific location of the output compressed bitstream in the output buffer.Depend on CABAC or CAVLC encoding mode (from Slice State), PAK Object Command is always ended in byte aligned output bitstream except for CABAC header insertion which is bit aligned. In the aligned cases, PAK will perform 0 filling in CAVLC mode, and 1 filling in CABAC mode.
Insertion data can include:any encoded syntax elements bit data before the encoded Slice Data (PAK Object Command) of the current SliceSPS NALPPS NALSEI NALOther Non-Slice NALLeading_Zero_8_bits (as many bytes as there is)Start Code PrefixNAL Header ByteSlice HeaderAny encoded syntax elements bit data after the encoded Slice Data (PAK Object Command) of the current Slice and prior to the next encoded Slice Data of the next Slice or prior to the end of the bistream, whichever comes firstCabac_Zero_Word or Trailing_Zero_8bits (as many bytes as there is).
Anything listed above before a Slice DataContext switch interrupt is not supported by this command.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="8">0</td><td>31:29</td><td>Command Type<table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>28:27</td><td>Pipeline<table><tr><td>Default Value:</td><td>2h MFX_PAK_INSERT_OBJECT</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>26:24</td><td>Media Command Opcode<table><tr><td>Default Value:</td><td>0h MFX_COMMON</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>23:21</td><td>SubOpcode A<table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>20:16</td><td>SubOpcode B<table><tr><td>Default Value:</td><td>8h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>15:12</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>11:0</td><td>DWord Length<table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1) = Variable Length in DW</td></tr><tr><td>Format:</td><td>=n Total Length - 2</td></tr></table></td></tr>
<tr><td rowspan="3">1</td><td>31:18</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>17:16</td><td>DataByteOffset - SrcDataStartingByteOffset[1:0]
Source Data Starting Byte Position within the very first inline DW.<table><tr><th>Programming Notes</th></tr><tr><td>Must be set to 0 for JPEG encoder</td></tr></table></td></tr>
<tr><td>15</td><td>HeaderLengthExcludeFrmSize
In case this flag is on, bits are NOT accumulated during current access unit coding neither for Cabac Zero Word insertion bits counting or for output in MMIO register MFC_BITSTREAM_BYTECOUNT_FRAME_NO_HEADER.
When using HeaderLenghtExcludeFrmSize for header insertion, the software needs to make sure that data comes already with inserted start code emulation bytes. SW shouldn't set EmulationFlag bit (Bit 3 of DWORD1 of MFX_PAK_INSERT_OBJECT).<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>NO_ACCUMULATION</td><td>Bits during current call are not accumulated</td></tr><tr><td>0</td><td>ACCUMULATE</td><td>All bits accumulated</td></tr></table></td></tr>
</table>

MFX_PAK_INSERT_OBJECT

Programming Notes
Must be set to 0 for JPEG encoder

14 **Slice Header Indicator**

This bit indicates if the insert object is a slice header.

Value	Name	Description
1	SLICE_HEADER	Insertion Object is a Slice Header. The command is stored internally by HW and is used for inserting slice headers.
0	LEGACY	Legacy Insertion Object command. The PAK Insertion Object command is not stored in HW.

Programming Notes
The payload for PAK_INS_OBJ should contain only start code for Slice header followed by NAL_type and slice header (slice_header() in AVC spec). The payload for PAK_INS_OBJ shouldn't contain CABAC Byte alignment bits. HW adds these alignment bits which are part of slice_data. Example PAK_INS_OBJ payload : 00 00 01 <NAL_type> <slice_header_Byte0> .....<slice_header_Byte LAST> Any zero_bytes that are added before slice header can be inserted by any preceding general PAK_INS_OBJ.

13:8 **DataBitsInLastDW - SrCDataEndingBitInclusion[5:0]**

Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first.For example, SrCDataEndingBitInclusion = 9, bit 7:0 and bit 15 are included as valid header data.

Value	Name
[1,32]	

7:4 **SkipEmulByteCnt - Skip Emulation Byte Count**

Skip emulation check for number of starting bytesIt can be programmed from 0 to 15 bytes.For example, to skip the start code that has already prefixed in the bitstream.

Programming Notes
Must be set to 0 for JPEG encoder

3 **EmulationFlag - EmulationByteBitsInsertEnable**

Value	Name	Description
0	NONE	No emulation
1	EMULATE	Instruct the hardware to perform Start Code Prefix (0x 00 00 01/02/03/00) Search and Prevention Byte (0x 03) insertion on the insertion data of this command. It is required that hardware will handle a start code prefix crossing the boundary between insertion commands, or an insertion command followed by a PAK Object command.

Programming Notes

MFX_PAK_INSERT_OBJECT

		Must be set to 0 for JPEG encoder
	2	**LastHeaderFlag - LastSrcHeaderDataInsertCommandFlag** To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series.In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.In CAVLC, hardware ignores this bit
	1	**EndOfSliceFlag - LastDstDataInsertCommandFlag** No more insertion command and no more PAK-OBJECT command follows.Flush data out to memory
	0	**BitstreamStartReset - ResetBitStreamStartingPos** Value / Name / Description: 1 / RESET / Reset the bitstream buffer insertion position to the bitstream buffer starting position. 0 / INSERT / Insert the current command inline data starting at the current bitstream buffer insertion position **Programming Notes** Must be set to 1 for JPEG encoder
2..n	31:0	**Insert Data PayLoad** Actual Data to be inserted to the output bitstream buffer.

MFX_PIPE_BUF_ADDR_STATE

MFX_PIPE_BUF_ADDR_STATE

Source: VideoCS

Length Bias: 2

This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers).

This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculated the corresponding memory location within the frame buffer directly.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_PIPE_BUF_ADDR_STATE Format: OpCode
	26:24	**Common Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Fixed Length Value: 3Fh — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)

MFX_PIPE_BUF_ADDR_STATE

1 — 31:6 — Pre Deblocking Destination Address

Format:	GraphicsAddress[31:6]

Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit).
This field is ignored if PreDeblockOutEnable is set to 0 (disable).

1 — 5:0 — Reserved

2 — 31:16 — Reserved

Format:	MBZ

2 — 15:0 — Pre Deblocking Destination Address High

Format:	GraphicsAddress[47:32]

This field is for the upper range of Pre-Deblocking Destination Address.
This field is ignored if **PreDeblockOutEnable** is set to 0 (disable).

3 — 31:15 — Reserved

Format:	MBZ

3 — 14:13 — Pre Deblocking - Tiled Resource Mode

For Media Surfaces:

This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

3 — 12:11 — Reserved

Format:	MBZ

3 — 10 — Pre Deblocking - Memory Compression Mode

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a**Memory Data Formats chapter -section**Media Memory Compression**for more details.**

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

Programming Notes
This bit is not used unless Memory Compression Enable is set to "1"
All Codec except JPEG Vertical Compression Only

MFX_PIPE_BUF_ADDR_STATE

Video Mode	Compression Mode
AVC Frame Only (No MBAFF or Field)	Vertical
VP8 (Only Frame is supported	Vertical

JPEG Decode Table

Chroma Format	Output Format	Compression Mode
422H_2Y, 422H_4Y	YUY2	Horizontal
422H_2Y, 422H_4Y	UYVY	Horizontal
422H_2Y, 422H_4Y	NV12	Not Supported
420	NV12	Vertical

9

Pre Deblocking - Memory Compression Enable

Format:	Enable

Memory compression will be attempted for this surface.

Value	Name
0	Compression Disable
1	Compression Enable

Programming Notes

Video Mode	Compression Enable
AVC Frame Only (No MBAFF or Field)	Yes
VP8 (Only Frame is supported	Yes

JPEG Decode

Chroma Format	Output Format	Compression Enable
422H_2Y,422H_4Y	YUY2	Yes
422H_2Y,422H_4Y	YUY2	Yes
422H_2Y,422H_4Y	UYVY	Yes
422H_2Y, 422H_4Y, 422V_2Y, 422V_4Y	NV12	No
420	YUY2, UYVY	No
420	NV12	Yes

8:7

Pre Deblocking - Arbitration Priority Control

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

MFX_PIPE_BUF_ADDR_STATE

	6:1	**Pre Deblocking - Index to Memory Object Control State (MOCS) Tables:** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
4	31:6	**Post Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit)This field is ignored if PostDeblockOutEnable is set to 0 (disable).
	5:0	**Reserved**
5	31:16	**Reserved** Format: MBZ
	15:0	**Post Deblocking Destination Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Post-Deblocking Destination Address. This field is ignored if **PostDeblockOutEnable** is set to 0 (disable).
6	31:15	**Reserved** Format: MBZ
	14:13	**Post Deblocking - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Post Deblocking - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats chapter -section** Media Memory Compression **for more details.**

Value	Name
0	Horizontal Compression Mode

MFX_PIPE_BUF_ADDR_STATE

		Programming Notes **This bit is not used unless Memory Compression Enable is set to "1"** **All Codec except JPEG Vertical Compression Only** Video Mode: AVC Frame Only (No MBAFF or Field) — Compression Mode: Horizontal Video Mode: VP8 (Only Frame is supported — Compression Mode: Horizontal Video Mode: JPEG — Compression Mode: Not Supported
	9	**Post Deblocking - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. Value 0: Compression Disable Value 1: Compression Enable **Programming Notes** Video Mode: AVC Frame Only (No MBAFF or Field) — Compression Enable: Yes Video Mode: VP8 (Only Frame is supported — Compression Enable: Yes Video Mode: JPEG — Compression Enable: No (In JPEG mode, this surface is not used)
	8:7	**Post Deblocking - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value 00b: Highest priority Value 01b: Second highest priority Value 10b: Third highest priority Value 11b: Lowest priority
	6:1	**Post Deblocking - Index to Memory Object Control State (MOCS) Tables:** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
7	31:6	**Original Uncompressed Picture Source Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned frame buffer address for fetching YUV pixel data from the original uncompressed input picture for encoding. This field is only valid in **encoding** mode.

MFX_PIPE_BUF_ADDR_STATE

	5:0	**Reserved** Format: MBZ
8	31:16	**Reserved** Format: MBZ
	15:0	**Original Uncompressed Picture Source Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Original Uncompressed Picture Source Address. This field is valid for **encoding** mode only.
9	31:15	**Reserved** Format: MBZ
	14:13	**Original Uncompressed Picture - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Original Uncompressed Picture - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

	9	**Original Uncompressed Picture - Memory Compression Enable** Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Compression Disable

MFX_PIPE_BUF_ADDR_STATE

	8:7	**Original Uncompressed Picture Source - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:1	**Original Uncompressed Picture - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
10	31:6	**StreamOut Data Destination Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned address for outputting the per-MB indirect data to memory when **StreamOutEnable** is set in the MFX_PIPE_MODE_SELECT command. For Decoder : This field is used for transcoding purpose. For Encoder : This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Format: MBZ
11	31:16	**Reserved** Format: MBZ
	15:0	**StreamOut Data Destination Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Original Uncompressed Picture Source Address
12	31:15	**Reserved** Format: MBZ
	14:13	**StreamOut Data Destination - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode. **Value** / **Name** / **Description** 0h – TRMODE_NONE – No tiled resource 1h – TRMODE_TILEYF – 4KB tiled resources 2h – TRMODE_TILEYS – 64KB tiled resources 3h – Reserved

MFX_PIPE_BUF_ADDR_STATE

	12:11	**Reserved** Format: MBZ
	10	**StreamOut Data Destination - Memory Compression Mode** Description: Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before Only horizontal mode is supported. Value 0: Horizontal Compression Mode Value 1: Vertical Compression Mode
	9	**StreamOut Data Destination - Memory Compression Enable** Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before. Value 0: Compression Disable
	8:7	**StreamOut Data Destination - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value 00b: Highest priority Value 01b: Second highest priority Value 10b: Third highest priority Value 11b: Lowest priority
	6:1	**StreamOut Data Destination - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
13	31:6	**Intra Row Store Scratch Buffer Base Address** Format: GraphicsAddress[31:6] This field provides the base address of the scratch buffer (read/write) used by the AVC/VP8 IntraPrediction unit to store MB information of the previous row for processing of each macroblock in the current row. The Intra Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of the current macroblock to address the Intra Row Store. This field is ignored in MPEG2 and VC1 mode.

MFX_PIPE_BUF_ADDR_STATE

Max 256 cachelines for 4K pixels (1 cacheline for either MBAFF or non-MBAFF)Intra Row Store Scratch Buffer - Arbitration Priority Control

Programming Notes

This is one of the four RowStore Scratch Buffers which can be programmed to use the internal Media Cache (total size 640 CacheLine). When Intra Row Store Scratch Buffer Cache Select is programmed to "1", this data will be stored inside MFX Media Internal Stroage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address to Media Cache. Driver needs to make sure the whole buffer fits into MFX Media Internal Storage. *(Notes: 1 cacheline per MB, and the buffer needs to have enough space for 1 MB row).*

	5:0	**Reserved** Format: MBZ
14	31:16	**Reserved** Format: MBZ
	15:0	**Intra Row Store Scratch Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Intra RowStore/Scratch Buffer Base Address This field is ignored in MPEG2 and VC1 mode.
15	31:15	**Reserved** Format: MBZ
	14:13	**Intra Row Store Scratch Buffer - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12 — **Intra Row Store Scratch Buffer Cache Select**

This field controls if Intra Row Store is going to store inside Media Cache or to LLC.

Value	Name	Description
0		Buffer going to LLC.
1		Buffer going to Internal Media Storage

11 — **Reserved**

Format: MBZ

10 — **Intra Row Store Scratch Buffer - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless

MFX_PIPE_BUF_ADDR_STATE

		Memory Compression Enable is set to "1" **Value** / **Name** 0 – Horizontal Compression Mode 1 – Vertical Compression Mode
	9	**Intra Row Store Scratch Buffer - Memory Compression Enable** **Value** / **Name** 0 – Compression Disable **Programming Notes** This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Intra Row Store Scratch Buffer - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:1	**Intra Row Store Scratch Buffer - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
16	31:6	**Deblocking Filter Row Store Scratch Base Address** Format: GraphicsAddress[31:6] Deblocking Filter Row Store is needed for: • AVC and VC1 In-Loop Deblocking Filter • VC1 Overlap-smoothing Filter • AVC, VC1, and MPEG-2 Out-Of-Loop Deblocking Filter (Intel extension) This field provides the 64 byte aligned base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address the Deblocking Filter Row Store. Max 6 cachelines for VC1 and MPEG2, and max 4 for AVC (for MBAFF, 2 for non-MBAFF) **Programming Notes**

MFX_PIPE_BUF_ADDR_STATE

		This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Cache (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage. *(Notes: 2 cachelines per MB for non-mbaff; 4 cahcelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).*
	5:0	**Reserved** Format: MBZ
17	31:16	**Reserved** Format: MBZ
	15:0	**Deblocking Filter Row Store Scratch Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Filter Row Store Scratch Buffer Address.
18	31:15	**Reserved** Format: MBZ
	14:13	**Deblocking Filter Row Store - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	12	**Deblocking Filter Row Store Scratch Buffer Cache Select** This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC. Value / Name / Description: 0 / / Buffer going to LLC 1 / / Buffer going to Media Internal Storage
	11	**Reserved** Format: MBZ
	10	**Deblocking Filter Row Store Scratch - Memory Compression Mode** Description: Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

MFX_PIPE_BUF_ADDR_STATE

		<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Horizontal Compression Mode</td></tr><tr><td>1</td><td>Vertical Compression Mode</td></tr></table>
	9	**Deblocking Filter Row Store Scratch - Memory Compression Enable** <table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table> <table><tr><th>Programming Notes</th></tr><tr><td>This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed</td></tr></table>
	8:7	**Deblocking Filter Row Store Scratch - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. <table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table>
	6:1	**Deblocking Filter Row Store Scratch - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
19..50	63:48	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
	47:32	**Reference Picture Address [n] High** <table><tr><td>Format:</td><td>Address[47:32]</td></tr></table> This field is for the upper range of Reference Picture Addresses
	31:6	**Reference Picture Address [n]** <table><tr><td>Format:</td><td>Address[31:6]</td></tr></table> Specifies the 64 byte aligned reference frame buffer addresses for the motion compensation operation in AVC/ /MPEG2. AVC can specify up to 16 YUV frame-based surfaces for both forward and backward references, i.e. L0+L1 total = 16 max. Any entry can be assigned to L0 or L1 or both lists.But VC1 and MPEG2, worst case, can use up to 2 YUV frame-based surfaces for both forward and backward references: • P-MB : RefAddr[0] - temporal closest previous field of a reference frame (can be the current frame)

MFX_PIPE_BUF_ADDR_STATE

- RefAddr[1]- next temporal closest previous field of a reference frame (must be different from the current frame)

It is a variant (without the LongTermRefPic specification) of the RefFrameList[16] defined in AVC DXVA Spec. RefAddr[0-15] is indexed by frame_storeID »1. It is not a packed list, i.e. invalid entries can scatter among the list. All invalid addresses must be set to a valid address RefAddr[0] by the driver. The same applies to VC1 and MPEG2.

Programming Notes
AVC: Always specifies all 16 addresses even some of them are not needed as indicated by the max num of active reference pictures. This is done for preventing data corruption (error, fault condition, etc.) by having all the references being set to a legal location.

	5:0	**Reserved** Format: MBZ
51	31:15	**Reserved** Format: MBZ
	14:13	**Reference Picture - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:9	**Reserved** Format: MBZ
	8:7	**Reference Picture - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

	6:1	**Reference Picture - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**

MFX_PIPE_BUF_ADDR_STATE

52	31:6	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address** Format: GraphicsAddress[31:6] **For decoder:** Specifies the 64 byte aligned buffer address for writing a single error/status record into the memory when **Pic Error/Status Report Enable** is set in the MFX_PIPE_MODE_SELECT Command. The error/status record is written by HW at the end of decoding one single picture. The record is written in a fixed format, total 96-bits in size always. Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format. **For encoder:** Specifies the 64 byte aligned buffer address for reading the per-MB indirect data from memory when **MacroblockStatEnable** is set in the MFX_AVC_IMG_STATE Command. This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit, and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Format: MBZ
53	31:16	**Reserved** Format: MBZ
	15:0	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Macroblock Status Buffer Base Address
54	31:15	**Reserved** Format: MBZ
	14:13	**Macroblock Status Buffer - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Macroblock Status Buffer - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

MFX_PIPE_BUF_ADDR_STATE

		Value / **Name** 0 – Horizontal Compression Mode 1 – Vertical Compression Mode
	9	**Macroblock Status Buffer - Memory Compression Enable** **Value** 0 – **Name** Compression Disable **Programming Notes** This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Macroblock Status Buffer - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:1	**Macroblock Status Buffer - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
55	31:6	**Macroblock ILDB StreamOut Buffer Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when **Debocker streamout enable** is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.
	5:0	**Reserved** Format: MBZ
56	31:16	**Reserved** Format: MBZ
	15:0	**Macroblock ILDB StreamOut Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Filter Row Store Scratch Address

MFX_PIPE_BUF_ADDR_STATE

57 | 31:15 Reserved

Format:	MBZ

14:13 Macroblock ILDB StreamOut - Tiled Resource Mode

For Media Surfaces:

This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12:11 Reserved

Format:	MBZ

10 Macroblock ILDB StreamOut Buffer - Memory Compression Mode

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

9 Macroblock ILDB StreamOut Buffer - Memory Compression Enable

Value	Name
0	Compression Disable

Programming Notes
This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed

8:7 Macroblock ILDB StreamOut Buffer - Arbitration Priority Control

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

MFX_PIPE_BUF_ADDR_STATE

	6:1	**Macroblock ILDB StreamOut Buffer - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
58	31:6	**Second Macroblock ILDB StreamOut Buffer Base Address** Format: GraphicsAddress[31:6] 64 byte aligned buffer. Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when Debocker streamout enable is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.
	5:0	**Reserved** Format: MBZ
59	31:16	**Reserved** Format: MBZ
	15:0	**Second Macroblock ILDB StreamOut Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Second Macroblock ILDB StreamOutBuffer Base Address.
60	31:15	**Reserved** Format: MBZ
	14:13	**Second Macroblock ILDB StreamOut Buffer - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Second Macroblock ILDB StreamOut Buffer - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

MFX_PIPE_BUF_ADDR_STATE

		Value / **Name** 0 – Horizontal Compression Mode 1 – Vertical Compression Mode
	9	**Second Macroblock ILDB StreamOut Buffer - Memory Compression Enable** **Value** / **Name** 0 – Compression Disable **Programming Notes** This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Second Macroblock ILDB StreamOut Buffer - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
	6:1	**Second Macroblock ILDB StreamOut Buffer - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
61	31	**Reference Picture 15 - Memory Compression Mode** **Description** Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before. **Value** / **Name** 0 – Horizontal Compression Mode 1 – Vertical Compression Mode
	30	**Reference Picture 15 - Memory Compression Enable** **Value** / **Name** 0 – Compression Disable 1 – Compression Enable
	29	**Reference Picture 14 - Memory Compression Mode** **Description** Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

MFX_PIPE_BUF_ADDR_STATE

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

28 **Reference Picture 14 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

27 **Reference Picture 13 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

26 **Reference Picture 13 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

25 **Reference Picture 12 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

24 **Reference Picture 12 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

MFX_PIPE_BUF_ADDR_STATE

23	**Reference Picture 11 - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

22	**Reference Picture 11 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

21	**Reference Picture 10 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

20	**Reference Picture 10 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

19	**Reference Picture 9 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

MFX_PIPE_BUF_ADDR_STATE

18 **Reference Picture 9 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

17 **Reference Picture 8 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

16 **Reference Picture 8 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

15 **Reference Picture 7 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

14 **Reference Picture 7 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

13 **Reference Picture 6 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

MFX_PIPE_BUF_ADDR_STATE

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

12 **Reference Picture 6 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

11 **Reference Picture 5 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

10 **Reference Picture 5 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

9 **Reference Picture 4 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

8 **Reference Picture 4 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

MFX_PIPE_BUF_ADDR_STATE

7 **Reference Picture 3 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

6 **Reference Picture 3 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

5 **Reference Picture 2 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

4 **Reference Picture 2 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

3 **Reference Picture 1 - Memory Compression Mode**

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

MFX_PIPE_BUF_ADDR_STATE

	2	**Reference Picture 1 - Memory Compression Enable** Value: 0 — Compression Disable Value: 1 — Compression Enable
	1	**Reference Picture 0 - Memory Compression Mode** Description: Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details. Note: This bit is not used unless Memory Compression Enable is set to "1" Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before. Value: 0 — Horizontal Compression Mode Value: 1 — Vertical Compression Mode
	0	**Reference Picture 0 - Memory Compression Enable** Value: 0 — Compression Disable Value: 1 — Compression Enable
62	31:6	**Scaled Reference Surface Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned down scaled reference frame buffer addresses that needs to be used by the PAK down-scaler to write the down scaled pixels. Only the luma pixels will be downscaled and written to the surface
	5:0	**Reserved** Format: MBZ
63	31:16	**Reserved** Format: MBZ
	15:0	**Scaled Reference Surface Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Scaled Reference Surface Base Address.
64	31:15	**Reserved** Format: MBZ
	14:13	**Scaled Reference Surface - Tiled Resource Mode** For Media Surfaces: This field specifies the tiled resource mode Value: 0h — Name: TRMODE_NONE — Description: No tiled resource

MFX_PIPE_BUF_ADDR_STATE

1h	TRMODE_TILEYF	No tiled resource
2h	TRMODE_TILEYS	No tiled resource
3h	Reserved	

12:11 Reserved

Format:	MBZ

10 Scaled Reference Surface - Memory Compression Mode

Description
Distinguishes Vertical from Horizontal compression. Please refer to vol1a Memory Data Formats chapter -section Media Memory Compression for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

9 Scaled Reference Surface - Memory Compression Enable

Format:	Enable

Memory compression shouldnt be enabled for this surface.

8:7 Scale Reference Surface - Arbitration Priority Control

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest Priority
01b	Second Highest Priority
10b	Third Highest Priority
11b	Lowest Priority

6:1 Scaled Reference Surface - Index to Memory Object Control State (MOCS) Tables

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers.

The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 Reserved

MFX_PIPE_MODE_SELECT

MFX_PIPE_MODE_SELECT

Source: VideoCS

Length Bias: 2

Specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis.

The MFX_PIPE_MODE_SELECT command specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis. It also configures the hardware pipeline according to the active encoder/decoder operating mode for encoding/decoding the current picture. Commands issued specifically for AVC and MPEG2 are ignored when VC1 is the active codec.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_COMMON Format: OpCode
	26:24	**Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpA** Default Value: 0h Format: OpCode
	20:16	**SubOpB** Default Value: 0h MFX_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 3h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31	**Reserved**
	30	**Reserved** Format: MBZ

MFX_PIPE_MODE_SELECT

	29	**Reserved**
	28:27	**Reserved**
	26	**Reserved** Format: MBZ
	25	**Reserved** Format: MBZ
	24	**Reserved** Format: MBZ
	23:19	**Reserved** Format: MBZ
	18	**Reserved**
	17	**Decoder Short Format Mode** For IT mode, this bit must be 0. **Value** / **Name** / **Description** 0 / Short Format Driver Interface **[Default]** / AVC/VC1/MVC/VP8 Short Format Mode is in use **Note: There is no Short Format for VP8 yet, so this field must be set to 1 for VP8.** 1 / Long Format Driver Interface / AVC/VC1/MVC/VP8 Long Format Mode is in use.
	16:15	**Decoder Mode select** Each coding standard supports two entry points: VLD entry point and IT (IDCT) entry point. This field selects which one is in use.This field is only valid if Codec Select is 0 (decoder). **Value** / **Name** / **Description** 0h / VLD Mode / All codec minimum must support this mode Configure the MFD Engine for VLD Mode Note: All codec minimum must support this mode 1h / IT Mode / Configure the MFD Engine for IT Mode Note: Only VC1 and MPEG2 support this mode 2h / Deblocker Mode / Configure the MFD Engine for Standalone Deblocker Mode. Require streamout AVC edge control information from preceeding decoding pass. 3h / Reserved /
	14	**Reserved**
	13	**Reserved**
	12	**Deblocker Stream-Out Enable** This field indicates if Deblocker information is going to be streamout during VLD decoding. For AVC, it is needed to enable the deblocker streamout as the AVC Disable_DLKFilterIdc is a slice level parameters. Driver needs to determine ahead of time if at least one slice of the current frame/ has deblocker ON.

MFX_PIPE_MODE_SELECT

		Value / **Name** / **Description** 0h / Disable / 1h / Enable /
	11	**Pic Error/Status Report Enable.** This field control whether the error/status reporting is enable or not.0: Disable1: EnableIn decoder modes: Error reporting is written out once per frame. The Error Report frame ID listed in DW3 along with the VLD/IT error status bits are packed into one cache and written to the "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command. Note: driver shall program different error buffer addresses between pictrues; otherwise, hardware might overwrite previous written data if driver does not read it fast enough.In encoder modes: Not used Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format. **Value** / **Name** 0h / Disable 1h / Enable
	10	**Stream-Out Enable** This field controls whether the macroblock parameter stream-out is enabled during VLD decoding for transcoding purpose. **Value** / **Name** 0h / Disable 1h / Enable **Programming Notes** In decoder modes: The Stream-Out feature is added to support transcoding. While decoding the input compressed stream, selected decoded information may be used by the encoder for re-compression.In encoder modes: This feature used to perform dynamic Multipass of PAK for conformance pupose. Also it provides feedback to host (ENC) for future needs. Software can use this bit to disable writing PAK steam data to the streamout buffer for last pass of frame in PAK. Thus, save memory bandwidth.
	9	**Post Deblocking Output Enable (PostDeblockOutEnable)** This field controls the output write for the reconstructed pixels AFTER the deblocking filter.In MPEG2 decoding mode, if this is enabled, VC1 deblocking filter is used. **Value** / **Name** 0h / Disable 1h / Enable

MFX_PIPE_MODE_SELECT

8 Pre Deblocking Output Enable (PreDeblockOutEnable)

This field controls the output write for the reconstructed pixels BEFORE the deblocking filter.

Value	Name
0h	Disable
1h	Enable

7 Scaled Surface Enable

This field indicates if the scaled surface is enabled. This field enables the 4x HME downscalar of the reconstructed image. Only supported for AVC and VP8 formats.

Value	Name
0h	Disable
1h	Enable

6 Frame Statistics StreamOut Enable

This field controls the frame level statistics streamout from the PAK.

Value	Name
0h	Disable
1h	Enable

5 Stitch Mode

Exists If:	//CodecSel=Encode and StandardSel=AVC

Value	Name	Description
0h	Not in stitch mode	
1h	In the special stitch mode	This mode can be used for any Codec as long as bitfield conditions are met.

4 Codec Select

Value	Name	Description
0h	Decode	
1h	Encode	Valid only if StandardSel is AVC and MPEG2)

3:0 Standard Select

Value	Name	Description
0000b	MPEG2	
0001b	VC1	
0010b	AVC	Covers both AVC and MVC
0011b	JPEG	
0101b	VP8	Decoder, Encoder
0110b	Reserved	
0111b	Reserved	
1111b	UVLD	SW decoder w/ embedded micro-controller and co-processor

MFX_PIPE_MODE_SELECT

2	31	**Reserved** Format: MBZ
	30	**Reserved**
	29	**Reserved** Format: MBZ
	28	**Reserved**
	27	**Reserved**
	26	**Reserved**
	25	**Reserved**
	24	**Reserved** Format: MBZ
	23	**Reserved**
	22:21	**Reserved**
	20:19	**Reserved** Format: MBZ
	18	**Reserved** Format: MBZ
	17	**Reserved**
	16	**Reserved**
	15	**Reserved**
	14	**VLF 720i (Odd Height) in VC1 Mode** This bit indicates VLF write out VC1 picture with odd height (in MBs). Value / Name / Description 0 / Disable **[Default]** / 1 / Enable / 720i Enable
	13	**Reserved** Format: MBZ
	12	**Reserved**
	11	**Reserved**
	10	**MPC pref08x8_disable Flag (Default 0)** Value / Name 0 / Disable 1 / Enable
	9	**Reserved** Format: MBZ

MFX_PIPE_MODE_SELECT

	8	**Reserved**
	7	**Reserved**
	6	**Clock gate Enable at Slice-level** BitFieldDesc: Value / Name / Description 0 / Disable / Disable Slice-level Clock gating, Unit-level Clock gating will apply 1 / Enable / Enable Slice-level Clock gating, overrides any Unit level Clock gating
	5	**Reserved**
	4	**Reserved** Format: MBZ
	3	**VDS ILDB Calculation** This bit forces all MB into INTRA MBs before doing ILDB control generation in VDS. Value / Name / Description 0 / Disable **[Default]** / Use original definition for ILDB calculation. 1 / Enable / Force neighbor Intra MB = 1 on ILDB BS calculation. **Programming Notes** When the bit is '0', the ILDB control generation will be the same as the original spec (AVC/VC1).
	2:1	**Reserved**
	0	**Reserved**
3	31:0	**Pic Status/Error Report ID** Exists If: //Decoder Mode Only Format: U32 In decoder modes: Error reporting is written out once per frame. This field along with the VLD error status bits are packed into one cache and written to the memory location specified by "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command. Value / Name / Description 0h / 32-bit unsigned / Unique ID Number 1h / Reserved /
4	31:0	**Reserved** Format: MBZ

MFX_QM_STATE

MFX_QM_STATE

Source: VideoCS

Length Bias: 2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices.This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B).In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 7h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Reserved** Format: MBZ
	1:0	**AVC** Exists If: //AVC- Decoder Only

MFX_QM_STATE

For AVC QM Type: This field specifies which Quantizer Matrix is loaded.

Value	Name
0	AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)
1	AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)
2	AVC_8x8_Intra_MATRIX
3	AVC_8x8_Inter_MATRIX

1:0 **MPEG2**

Exists If:	//MPEG2- Decoder Only

For MPEG2 QM Type: This field specifies which Quantizer Matrix is loaded.

Value	Name
0	MPEG_INTRA_QUANTIZER_MATRIX
1	MPEG_NON_INTRA_QUANTIZER_MATRIX
2-3	Reserved

1:0 **JPEG**

Exists If:	//JPEG- Encoder Only

For JPEG QM Type:This field specifies which Quantizer Matrix is loaded.

Value	Name
0	JPEG_Luma_Y_QUANTIZER_MATRIX (or R)
1	JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G)
2	JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B)

Programming Notes
For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j]. In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.

2..33 31:0 **Forward Quantizer Matrix**

Format:	U32

The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.

MFX_STATE_POINTER

MFX_STATE_POINTER

Source: VideoCS

Length Bias: 2

The MFX_STATE_POINTER command, issued at picture level, is used to set up the indirect pointers for VCS to fetch all the MFX states (Image state, Slice state, etc.) needed for the encoding/decoding process in PAK/IT mode. The encoding/decoding states are presented by state commands, which are grouped into separate sets (picture level, slice level, etc.), and each is stored in its own memory buffer referred by an indirect state pointer. The content of each indirect state buffer is a list of MFX state commands with no special format requirements. The sequence of commands in each indirect state buffer is terminated by a MI_BATCH_BUFFER_END command (acts as the last command marker). Therefore, indirect state buffers can have different and variable length of command sequences.

The indirection is designed to facilitate context switching in the middle of a codec operation. The smallest granularity of interruption is designed to be at a completed MB row in AVC/VC1/MPEG2 IT and AVC PAK operating modes as well as in VC1/MPEG2 VLD mode. There is no support for context switch in AVC VLD mode.

Hardware supports up to 4 separate indirect state pointers, allowing software to manage the grouping of state commands. During context switch, hardware restores (re-issues) the latest version of each indirect state pointer, if present.

MFX_STATE_POINTER command can only program one indirect state pointer at a time. MI_FLUSH will invalidate all indirect state buffer pointers inside VCS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFX_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved** Format: MBZ

MFX_STATE_POINTER

	11:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n Total Length - 2
1	31:5	**State Pointer** Format: GeneralStateOffset[31:5]Indirect State Buffer Specifies the 32-byte aligned address of an Indirect State Buffer. This pointer is relative to the General State Base Address.
	4:2	**Reserved** Format: MBZ
	1:0	**State Pointer Index** Specifies one of the four indirect state pointers to program.

Value	Name	Description
00b		indirect state pointer 0 (image state)
01b		indirect state pointer 1 (slice state)sc
10b		indirect state pointer 2
11b		indirect state pointer 3

MFX_STITCH_OBJECT

MFX_STITCH_OBJECT

Source: VideoCS

Length Bias: 2

The MFC_STITCH_OBJECT command is used when stitch-enabled is set to 1, while CodecSel and StandardSel are set to ENCODE and AVC, respectively. This command is used, for example, to stitch multiple bitstreams to form a transport stream.
It is a variable length command as the data to be inserted are presented as either inline data and/or indirect data of this command. Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid output. Hardware keeps track of an output bitstream buffer current byte position and the associated next bit insertion position index. Context switch interrupt is not supported by this command.

The input bitstreams may be elementary streams stored in protected memory.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_STITCH_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: Ah Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = Variable Length in DW (>= 3) Format: =n Total Length - 2 If it is 3, it indicates the absent of inline data.

MFX_STITCH_OBJECT

1	31:18	**Reserved** Format: MBZ
	17:16	**Source Data Starting Byte Offset** Source Data Starting Byte Position within the very first inline DW.
	15:14	**Reserved** Format: MBZ
	13:8	**Source Data Ending Bit Inclusion** Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first.For example, SrCDataEndingBitInclusion =9, bit 7:0 and bit 15 are included as valid header data. Value: [1,32] — Name:
	7:4	**Reserved**
	3	**Reserved**
	2	**Last Source Header Data Insert Command Flag** To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series.In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.In CAVLC, hardware ignores this bit.
	1	**Last Destination Data Insert Command Flag** THIS FIELD MUST BE THE SAME AS Last Source Header Data Insert Command Flag No more insertion command and no more PAK-OBJECT command follows.Flush data out to memory
	0	**Reserved**
2	31:19	**Reserved** Format: MBZ
	18:0	**Indirect Data Length** Format: U19 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.
3	31:0	**Indirect Data Start Address** Format: MfxIndirectBitstreamObjectAddress[31:0] This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the MFX Indirect Bitstream Object Base Address. Hardware ignores this field if indirect data is not present.
4..n	31:0	**Insert Data PayLoad** Inline data to be inserted to the output bitstream buffer

MFX_SURFACE_STATE

MFX_SURFACE_STATE

Source:	VideoCS
Length Bias:	2

This command is common for all encoding/decoding modes, to specify the uncompressed YUV picture (i.e. destination surface) or intermediate streamout in/out surface (e.g. coefficient/residual) (field, frame or interleaved frame) format for reading and writing:

- Uncompressed, original input picture to be encoded
- Reconstructed non-filtered/filtered display picturec(becoming reference pictures as well for subsequent temporal inter-prediction)

Since there is only one media surface state being active during the entire encoding/decoding process, all the uncompressed/reconstructed pictures are defined to have the same surface state. The primary difference among picture surface states is their individual programmed base addresses, which are provided by other state commands and not included in this command. MFX engine is making the association of surface states and corresponding buffer base addresses.
MFX engine currently supports only one media surface type for video and that is the NV12 (Planar YUV420 with interleaved U (Cb) and V (Cr). For optimizing memory efficiency based on access patterns, only TileY is supported. For JPEG decoder, only IMC1 and IMC3 are supported. Pitch can be wider than the Picture Width in pixels and garbage will be there at the end of each line. The following describes all the different formats that are supported and not supported in Gen7 MFX :

- NV12 - 4:2:0 only; UV interleaved; Full Pitch, U and V offset is set to 0 (the only format supported for video codec); vertical UV offset is MB aligned; UV xoffsets = 0. JPEG does not support NV12 format because non-interleave JPEG has performance issue with partial write (in interleaved UV format)
- IMC 1 & 3 - Full Pitch, U and V are separate plane; (JPEG only; U plane + garbage first in full pitch followed by V plane + garbage in full pitch). U and V vertical offsets are block aligned; U and V xoffset = 0; there is no gap between Y, U and V planes. IMC1 and IMC3 are different by a swap of U and V. This is the only format supported in JPEG for all video subsampling types (4:4:4, 4:2:2 and 4:2:0)
- We are not supporting IMC 2 & 4 - Full Pitch, U and V are separate plane (JPEG only; U plane first in full pitch followed by V plane in full pitch - U and V plane are side-by-side). U and V vertical offsets are 16-pixel aligned; V xoffset is half-pitch aligned; U xoffset is 0; there is no gap between Y, U and V planes. IMC2 and IMC4 are different by a swap of U and V.
- We are not supporting YV12 - half pitch for each U and V plane, and separate planes for Y, U and V (U plane first in half pitch followed by V plane in half pitch). For YV12, U and V vertical offsets are block aligned; U and V xoffset = 0; there is no gap between Y, U and V planes

Note that the following data structures are not specified through the media surface state

- 1D buffers for row-store and other miscellaneous information.
- 2D buffers for per-MB data-structures (e.g. DMV biffer, MB info record, ILDB Control and Tcoeff/Stocoeff).

This surface state here is identical to the Surface State for deinterlace and sample_8x8 messages described in the Shared Function Volume and Sampler Chapter.
For non pixel data, such as row stores, indirect data (Compressed Slice Data, AVC MV record, Coeff record and

MFX_SURFACE_STATE

AVC ILDB record) and streamin/out and output compressed bitstream, a linear buffer is employed. For row stores, the H/W is designed to guarantee legal memory accesses (read and write). For the remaining cases, indirect object base address, indirect object address upper bound, object data start address (offset) and object data length are used to fully specified their corresponding buffer. This mechanism is chosen over the pixel surface type because of their variable record sizes.
All row store surfaces are linear surface. Their addresses are programmed in Pipe_Buf_Base_State or Bsp_Buf_Base_Addr_State

Programming Notes

VC1 I picture scaling: Even though VC1 allows I reconstructed picture scaling (via RESPIC), as such scaling is only allowed at I picture. All subsequent P (and B) pictures must have the same picture dimensions with the preceding I picture. Therefore, all reference pictures for P or B picture can share the same surface state with the current P and B picture. Note : H/W is not processing RESPIC. Application is no longer expecting intel decoder pipelineand kernel to perform this function, it is going to be done in the video post-processing scaler or display controller scale as a separate step and controller.

All video codec surfaces must be NV12 Compliant, except JPEG. U/V vertical must be MB aligned for all video codec (further contrained for field picture), but JPEG can be block aligned. All video codec and JPEG uses Tiled - Y format only, for uncompressed pixel surfaces.

Even for JPEG planar 420 surface, application may provide only 1 buffers, but there is still only one single surface state for all of them. If IMC equal to 1, 2, 3 or 4, U and V have the pitch same as Y. And U and V will have different offset, each offset is block aligned.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h PARALLEL_VIDEO_PIPE
		Format:	OpCode
	28:27	**Pipeline**	
		Default Value:	2h MFX_COMMON
		Format:	OpCode
	26:24	**Opcode**	
		Default Value:	0h MFX_COMMON_STATE
		Format:	OpCode
	23:21	**SubOpA**	
		Default Value:	0h
		Format:	OpCode
	20:16	**SubOpB**	
		Default Value:	1h
		Format:	OpCode
	15:12	**Reserved**	
		Format:	MBZ

MFX_SURFACE_STATE

	11:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 4h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:4	**Reserved** Format: MBZ
	3:0	**Surface Id** Format: U4 **Value** / **Name** / **Description** 0000b / Reserved / 0001b / Reserved / 0010b / Reserved / 0011b / Reserved / 0100b / Source Input Picture (encoder) / 8-bit uncompressed data 0101b / Reconstructed Scaled Reference Picture / 8-bit data
2	31:18	**Height** Format: U14-1 Height This field specifies the height of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. Note : Gen7 Video Codecs must program less than and equal to 4K.(In future, it will be ideal to have this field define in a WORD boundary.)AVC - multiple of 2 MB rows for field pictureVC1 - mulitple of 4 pixels for field pictureMPEG2 - multiple of 2 MB rows for field picJPEG - mulitple of integral MCU (8 or 16 pixels) per picture **Value** / **Name** / **Description** [0,16383] / / representing heights [1,16384] **Programming Notes** • For AVC : For frame picture is a multiple of 16; for field picture is a multiple of 32 • For VC1 : For progressive frames, the frame height and frame width is a multiple of 2 pixels. For interlaced frames, the frame height shall be a multiple of 4 pixels, and its width is a multiple of 2 pixels, based on a PLANAR_420 surface.

MFX_SURFACE_STATE

17:4 Width

Format:	U14-1 Width

This field specifies the width of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the width of the Y (luma) plane.

Value	Name	Description
[0,16383]		representing widths [1,16384]

Programming Notes

- The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field).
- Width (field value + 1) must be a multiple of 2 for PLANAR_420,
- MFX HW does not use this field, the picture width is read from IMG State instead, because this field may not equal to the actual picture width. This field is used by the KMD to allocate surface in GTT.

3:2 Reserved

Format:	MBZ

1:0 Cr(V)/Cb(U) Pixel Offset V Direction

Format:	U0.2 exactly as shown in the original spec

Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction

Programming Notes

This field is ignored for all formats except PLANAR_420_8

3 31:28 Surface Format

Format:	U4

Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.Usage: For 420 planar YUV surface, use 4; for monochrome surfaces, use 12. For monochrome surfaces, hardware ignores control fields for Chroma planes.This field must be set to 4 - PLANAR_420_8, or 12 - Y8_UNORMNot used for MFX, and is ignored. But for JPEG decoding, this field should be programmed to the same format as JPEG_PIC_STATE. For video codec, it should set to 4 always.

Value	Name	Description
0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	(NV12, IMC1,2,3,4, YV12)
5	PLANAR_411_8	Deinterlace Only
6	PLANAR_422_8	Deinterlace Only

MFX_SURFACE_STATE

7	STMM_DN_STATISTICS	Deinterlace Only
8	R10G10B10A2_UNORM	Sample_8x8 Only
9	R8G8B8A8_UNORM	Sample_8x8 Only
10	R8B8_UNORM (CrCb	Sample_8x8 Only
11	R8_UNORM (Cr/Cb)	Sample_8x8 Only
12	Y8_UNORM	Sample_8x8 Only

27 **Interleave Chroma**

Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.For AVC/VC1/MPEG VLD and IT modes : set to Enable to support interleave U/V only.For JPEG : set to Disable for all formats (including 4:2:0) - because JPEG does not support NV12. (This field is needed only if JPEG will support NV12; otherwise is ignored.)

Value	Name
1	Enable
0	Disable

26:20 **Reserved**

Format:	MBZ

19:3 **Surface Pitch**

Format:	U17-1 pitch in Bytes

This field specifies the surface pitch in (#Bytes).

Value	Name	Description
[0,2047]		to [1B, 2048B]

Programming Notes

For tiled surfaces, the pitch must be a multiple of the tile width (i.e.128 bytes aligned). If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.For Y-tiled surfaces: Range = [127, 524287] to [128B,256KB] = [1 tile, 2048 tiles]

If Media Memory Compression is enabled, the following max pitch size restriction must be honored. For larger resolution, Media Memory compression Must be disabled.

Tiling Mode	Pixel Format	Max Frame Width (bytes)	Max Frame Width (pixels)	Max Pitch (bytes)
Legacy 4K	8bpp	16k	16k	16k + 127
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 127
	128bpp	16k	1k	16k + 127

MFX_SURFACE_STATE

TileYF	8bpp	8k	8k	8k + 63
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 255
	128bpp	16k	1k	16k + 255
TileYS	8bpp	16k	16k	16k + 255
	16bpp	16k	8k	16k + 511
	32bpp	16k	4k	16k + 511
	64bpp	16k	2k	16k + 1023
	128bpp	16k	1k	16k + 1023

2 **Half Pitch for Chroma**

Format:	Enable

(This field must be set to Disable)This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field. This field is only used for PLANAR surface formats.This field is igored by MFX (unless we support YV12)

1 **Tiled Surface**

Format:	Boolean

(This field must be set to TRUE: Tiled)This field specifies whether the surface is tiled.This field is ignored by MFX

Value	Name	Description
0	False	Linear
1	True	Tiled

Programming Notes
Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory.The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.

0 **Tile Walk**

Format:	3D_Tilewalk

(This field must be set to 1: TILEWALK_YMAJOR)This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See Memory Interface Functions for details on memory tiling and restrictions.This field is ignored when the surface is linear.This field is ignored by MFX. Internally H/W is always treated this set to 1 for all video codec and for JPEG.

Value	Name	Description
0h	XMAJOR	TILEWALK_XMAJOR
1h	YMAJOR	TILEWALK_YMAJOR

MFX_SURFACE_STATE

		Programming Notes The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit
4	31	**Reserved** Format: MBZ
	30:16	**X Offset for U(Cb)** Format: U15 Pixel Offset This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. This field must be set to zero.X Offset for U(Cb) in pixel (This field must be zero for NV12 and IMC 1 and 3) **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be zero.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for U(Cb)** Format: U15 Pixel Row Offset This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs. For JPEG, this field must be a multiple of 16 pixels.
5	31:29	**Reserved** Format: MBZ
	28:16	**X Offset for V(Cr)** Format: U13 Offset in Pixels This field must be zero for NV12 and IMC 1 and 3 This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.

MFX_SURFACE_STATE		
	15:0	**Y Offset for V(Cr)** Format: U16 Row Offset in Pixels This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. This field is ignored by all video codec, only used by JPEG. **Programming Notes** For PLANAR_420 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs. For JPEG, this field must be a multiple of 16 pixels.

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_DIRECTMODE_STATE

Source: VideoCS
Length Bias: 2
Exists If: //VC1 decoding in VLD modes

This is a picture level command and should be issued only once, even for a multi-slices picture.
There is only one DMV buffer for read (when processing a B-picture) and one for write (when processing a P-Picture). Each DMV record is 64 bits per MB, to store the top and bottom field MVs (32-bit MVx,y each).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VC1_DIRECTMODE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0005h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:6	**Direct MV Write Buffer Base Address for the Current Picture** This field provides the base address of the DMV write buffer to store the motion vectors decoded in the current picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software.This buffer must be 64-byte cacheline aligned.The write buffer size is 557,056 bytes for 1 frame. Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution).This field is only valid for a P picture

MFX_VC1_DIRECTMODE_STATE

	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Direct MV Write Buffer Base Address for the Current Picture [47:32]** This field is for the upper range of Direct MV Write Buffer Base Address for the Current Picture. This field is used for 48-bit addressing.
3	31:15	**Reserved** Format: MBZ
	14:13	**Direct MV Write Buffer - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ
	10	**Direct MV Write Buffer - Memory Compression Mode** Format: U1 Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

	9	**Direct MV Write Buffer - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface.
	8:7	**Direct MV Write Buffer Base Address for the Current Picture - Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority

MFX_VC1_DIRECTMODE_STATE

		10b: Third highest priority 11b: Lowest priority
	6:1	**Direct MV Write Buffer - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
4	31:6	**Direct MV Read Buffer Base Address for the Reference Picture** This field provides the base address of the DMV buffer for reference picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software.All these buffers must be 64-byte cacheline aligned.This field is only valid for a B picture.
	5:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ Reserved for 64-bit address extension.
	15:0	**Direct MV Read Buffer Base Address for the Current Picture [47:32]** This field is for the upper range of Direct MV Read Buffer Base Address for the Current Picture. This field is used for 48-bit addressing.
6	31:15	**Reserved** Format: MBZ
	14:13	**Direct MV Read Buffer - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Format: MBZ

MFX_VC1_DIRECTMODE_STATE

10 — Direct MV Read Buffer - Memory Compression Mode

Format:	U1

Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

9 — Direct MV Read Buffer - Memory Compression Enable

Format:	Enable

Memory compression will be attempted for this surface.

8:7 — Direct MV Read Buffer Base Address for the Current Picture - Arbitration Priority Control

Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:1 — Direct MV Read Buffer - Index to Memory Object Control State (MOCS) Tables

Format:	U6

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 — Reserved

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_PRED_PIPE_STATE

Source: VideoCS

Length Bias: 2

This command is used to set the operating states of the MFD Engine beyond the BSD unit. It is used with both VC1 Long and Short format.Driver is responsible to take the intensity compensation enable signal, the LumScale and the LumShift provided from the DXVA2 VC1 interface, and maintain a history of these values for reference pictures. Together with these three parameters specified for the current picture being decoded, driver will derive and supply the above sets of LumScaleX, LumShiftX and intensity compensation enable (single or double, forward or backward) signals. H/W is responsible to take these state values, and use them to build the lookup table (including the derivation of iScale and iShift) for remapping the reference frame pixels, as well as perfoming the actual pixel remapping calculations/process.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VC1_PRED_PIPE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0004h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:16	**Reserved** Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	15:14	**vin_intensitycomp_Double_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	13:12	**vin_intensitycomp_Double_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	11:10	**vin_intensitycomp_Single_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	9:8	**vin_intensitycomp_Single_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:4	**Reference Frame Boundary Replication Mode** Format: U4 This is a bit field with each bit indicating the corresponding picture's boundary replication mode. Bit 11: reference 3 Bit 10: reference 2 Bit 9: reference 1 Bit 8: reference 0 0 = progressive frame replication 1 = interlace frame replication This field is maintained and provided by driver for both long and short VC1 interface format.

MFX_VC1_PRED_PIPE_STATE		
	3:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ
	29:24	**LumShift2- single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - Single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
3	31:30	**Reserved** Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - double -FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
4	31:30	**Reserved** Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - Single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
5	31:30	**Reserved** Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - double -BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

Source: VideoCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 3h MFX_VP8_BSP_BUF_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Value / Name / Description 000h / Excludes DWord (0,1) **[Default]** / A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 008h / / Used for normal encode mode
1	31:6	**Frame Header Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format. 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Frame Header Base Addr - Upper Range**
3	31:9	**Reserved** Format: MBZ
	8:7	**Frame Header Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**for FrameHeaderBaseAddr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** / **Description** 00b / Cacheable / Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01b / UC / Uncacheable - non-cacheable 10b / WT / Writethrough 11b / WB / Writeback
	4:3	**Frame Header Base Addr - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching **Value** / **Name** 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	1:0	**Frame Header Base Addr - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. **Value** / **Name** 00b – Good chance of generating hits 01b – Next good chance of generating hits 10b – Decent chance of generating hits 11b – Poor chance of generating hits
4	31:6	**Intermediate Buffer Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:0	**Intermediate Buffer Base Addr - Upper Range**
6	31:9	**Reserved** Format: MBZ
	8:7	**Intermediate Buffer Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	6:5	**Intermediate Buffer Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Cacheable	Use Cacheability Controls from page table / UC with Fence (if coherent cycle)
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

	4:3	**Intermediate Buffer Base Addr- Target Cache (TC)** This field allows the choice of LLC vs. eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

	2	**Reserved**
	1:0	**Intermediate Buffer Base Addr- Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
00b	Good chance of generating hits
01b	Next good chance of generating hits
10b	Decent chance of generating hits
11b	Poor chance of generating hits

7	31:0	**Intermediate Buffer Partition-1 Offset**

Format:	U32

Programming Notes
All **Intermediate Buffer Partition-[i] Offset** (i = 1 to 8) and **Intermediate Buffer Max Size** need to be cacheline aligned (64Byte aligned).

8	31:0	**Intermediate Buffer Partition-2 Offset**

Format:	U32

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

9	31:0	**Intermediate Buffer Partition-3 Offset** Format: U32
10	31:0	**Intermediate Buffer Partition-4 Offset** Format: U32
11	31:0	**Intermediate Buffer Partition-5 Offset** Format: U32
12	31:0	**Intermediate Buffer Partition-6 Offset** Format: U32
13	31:0	**Intermediate Buffer Partition-7 Offset** Format: U32
14	31:0	**Intermediate Buffer Partition-8 Offset** Format: U32
15	31:0	**Intermediate Buffer Max Size** Format: U32
16	31:6	**Final Frame Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:0	**Reserved** Format: MBZ
17	31:16	**Reserved** Format: MBZ
	15:0	**Final Frame Base Addr - Upper Range**
18	31:9	**Reserved** Format: MBZ
	8:7	**Final Frame Base Addr - Arbitration Priority Control**

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	6:5	**Final Frame Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Cacheable	Use Cacheability Controls from page table / UC with Fence (if coherent cycle)
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

	4:3	**Final Frame Base Addr - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

	2	**Reserved**
	1:0	**Final Frame Base Addr - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
00b	Good chance of generating hits
01b	Next good chance of generating hits
10b	Decent chance of generating hits
11b	Poor chance of generating hits

19	31:6	**Reserved**

Format:	MBZ

	5:0	**FinalFrameByteOffset**

Format:	U6

Specify byte offset within a 64-byte cacheline where the bitstream should be inserted at.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

20	31:6	**Streamout Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Streamout Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b – Highest priority Value: 01b – Second highest priority Value: 10b – Third highest priority Value: 11b – Lowest priority
	3	**Reserved**
	2	**Streamout Base Addr - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Streamout Base Addr - Cacheability Control** Format: U2 EnumeratedType This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC) 00b – GTT Entry – Use cacheability control bits from GTT entry 01b – Not LLC or MLC – Data is not cached in LLC or MLC 10b – LLC but not MLC – Data is cached in LLC but not MLC 11b – Both LLC and MLC – Data is cached in both LLC and MLC
21	31:16	**Reserved** Format: MBZ
	15:0	**Streamout Base Addr - Upper Range**
22	31:9	**Reserved** Format: MBZ

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	8:7	**Streamout Base Addr - Arbitration Priority Control** Value / Name 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Streamout Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. Value / Name / Description 00b: Cacheable: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01b: UC: Uncacheable - non-cacheable 10b: WT: Writethrough 11b: WB: Writeback
	4:3	**Streamout Base Addr - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching Value / Name 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Streamout Base Addr - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. Value / Name 00b: Good chance of generating hits 01b: Next good chance of generating hits 10b: Decent chance of generating hits 11b: Poor chance of generating hits

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

23	31:6	**Coeff Probs StreamIn Surface** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Coeff Probs StreamIn Surface - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b – Highest priority; 01b – Second highest priority; 10b – Third highest priority; 11b – Lowest priority
	3	**Reserved**
	2	**Coeff Probs StreamIn Surface - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Coeff Probs StreamIn Surface - Cacheability Control** Format: U2 EnumeratedType This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC) 00b – GTT Entry – Use cacheability control bits from GTT entry 01b – Not LLC or MLC – Data is not cached in LLC or MLC 10b – LLC but not MLC – Data is cached in LLC but not MLC 11b – Both LLC and MLC – Data is cached in both LLC and MLC
24	31:16	**Reserved** Format: MBZ
	15:0	**Coeff Probs StreamIn Surface - Upper Range** This field is for the upper range of Coeff Probs[4][8][3][11]
25	31:9	**Reserved** Format: MBZ

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

8:7	**Coeff Probs StreamIn Surface - Arbitration Priority Control** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
6:5	**Coeff Probs StreamIn Surface - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. 00b: Cacheable - Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01b: UC - Uncacheable - non-cacheable 10b: WT - Writethrough 11b: WB - Writeback
4:3	**Coeff Probs StreamIn Surface - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
2	**Reserved**
1:0	**Coeff Probs StreamIn Surface - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 00b: Good chance of generating hits 01b: Next good chance of generating hits 10b: Decent chance of generating hits 11b: Poor chance of generating hits

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

26	31:6	**Token Statistics Surface** Format: TokenStatisticsAddress[31:6] 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Frame Header Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b – Highest priority; 01b – Second highest priority; 10b – Third highest priority; 11b – Lowest priority
	3	**Reserved**
	2	**Token Statistics Surface - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Token Statistics Surface - Cacheability Control** This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC). 00b – GTT Entry – Use cacheability control bits from GTT entry 01b – Not LLC or MLC – Data is not cached in LLC or MLC 10b – LLC but not MLC – Data is cached in LLC but not MLC 11b – Both LLC and MLC – Data is cached in both LLC and MLC
27	31:16	**Reserved** Format: MBZ
	15:0	**Token Statistics Surface** This field is for the upper range of Token Statistics Address.
28	31:9	**Reserved** Format: MBZ

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	8:7	**Token Statistics Surface - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for CoeffProbs StreamIn Surface** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** / **Description** 00b – Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01b – UC – Uncacheable - non-cacheable 10b – WT – Writethrough 11b – WB – Writeback
	4:3	**Token Statistics Surface - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching. **Value** / **Name** 00b – eLLC Only - not snooped in GT 01b – LLC Only 10b – LLC/eLLC Allowed 11b – L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Token Statistics Surface - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. **Value** / **Name** 11b – Good chance of generating hits. 10b – Next good chance of generating hits 01b – Decent chance of generating hits 00b – Poor chance of generating hits

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

29..30	63:6	**MPC RowStore Surface Address Low** Format: GraphicsAddress63-6 Abs. Address StreamIn/StreamOut Surface. **Note:** The format is linear vs. tile for better performance. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.
	5:4	**MPC RowStore Base Addr - Arbitration Priority Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value: Name 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	3	**Reserved**
	2	**MPC RowStore Surface Graphics Data Type (GFDT)** Format: U1 **This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT.** **The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.**
	1:0	**MPC RowStore Surface - Cacheability Control** **This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC).** Value: Name: Description 00b: GTT Entry: Use cacheability control bits from GTT entry 01b: Not LLC or MLC: Data is not cached in LLC or MLC 10b: LLC but not MLC: Data is cached in LLC but not MLC 11b: Both LLC and MLC: Data is cached in both LLC and MLC
31	31:9	**Reserved** Format: MBZ
	8:7	**MPC RowStore - Arbitration Priority Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value: Name 00b: Highest Priority 01b: Second highest Priority 10b: Third highest Priority 11b: Lowest Priority

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	6:5	**MPC RowStore - Memory Type: LLC/eLLC Cacheability Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** **Value** / **Name** / **Description** 00b — Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01b — UC — Uncacheable - non-cacheable 10b — WT — Writethrough 11b — WB — Writeback
	4:3	**MPC RowStore - Target Cache** Format: U2 **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** **Value** / **Name** 00b — eLLC Only 01b — LLC Only 10b — LLC/eLLC Allowed 11b — L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**MPC RowStore Surface - Age for QUADLRU (AGE)** **This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0").** **This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.** **Value** / **Name** 11b — Good chance of generating hits. 10b — Next good chance of generating hits. 01b — Decent chance of generating hits. 00b — Poor chance of generating hits.

MFX_VP8_Encoder_CFG

MFX_VP8_Encoder_CFG

Source: VideoCS

Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_PIC_STATE.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 1h MFX_VP8_ENCODER_CFG Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Value / Name / Description: 000h / Excludes DWord (0,1) **[Default]** / A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 01Dh / / Used for normal encode mode
1	31:11	**Reserved** Format: MBZ

MFX_VP8_Encoder_CFG

	10	**VBSPunitPowerClock Gating Disable** Format: U1 VBSPunit Power Clock Gating Disable.
	9	**Compressed Bitstream Output Disable** Format: U1 Disable Compressed Bitstream Output.**(Both Final Bitstream and Intermediate bit buffer)**
	8	**Finer BRC Enable** Format: U1 Enable Finer BRC Feature.
	7	**Per Segment Delta Qindex / LoopFilter Disable** Format: U1 Disable Per Segment Delta Qindex / Loop Filter in Rate Control.
	6	**Rate Control Initial Pass** Format: U1 Value / Name: 1 – Initial pass 0 – Subsequence Pass(es)
	5	**Skip Final Bitstream when Over / Under flow** Format: U1 Skip Final Bitstream conditionally on Over/Under flow in rate control and intermediate Bit Buffer Overrun.
	4	**Update Segment Feature Data Flag** Exists If: //VP8 Encoder Format: U1 Enable for Frame Header per Segment Quantizer / LoopFilter Update
	3	**Bitstream Statistics Output Enable** Enable Bitstream Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]
	2	**Token Statistics Output Enable** Enable Token Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]

MFX_VP8_Encoder_CFG

	1	**Final Bitstream Output Disable** Format: U1 Disable Final Bitstream Output.
	0	**Performance Counter Enable** Format: U1 Enable Performance Counter in Streamout.
2	31:8	**Reserved** Format: MBZ
	7	**Qindex_Clamp_High_mask for overflow** Format: U1 If current frame is overflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit1. In another word, subsequent passes would be skipped.
	6	**Qindex_Clamp_High_mask for underflow** Format: U1 If current frame is underflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit0. In another word, subsequent passes would be skipped
	5	**Final Bistream Buffer Overrun Enable Mask** Format: U1 Enable Final Bitstream Buffer Overrun detection feature.
	4	**Intermediate Bit Buffer Overrun Enable Mask** Format: U1 Enable Intermediate Bit Buffer Overrun detection feature.
	3	**Max Intra MB Bit Count Check Enable Mask** Format: U1 Enable Max. Intra MB bit count check in Streamout.
	2	**Max Inter MB Bit Count Check Enable Mask** Format: U1 Enable Max. Inter MB bit count check in Streamout.

MFX_VP8_Encoder_CFG

	1	**Min Frame Bit Count Rate Control Enable Mask** Format: U1 Enable Min. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is less than or equal to FrameBitRateMin. **Value** / **Name** / **Description** 1 / / If (Total Frame Level Bit Counter) =< (Frame Bit Rate Minimum limit) Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS Control Register. 0 / / Do not update bit[0] of MFX_VP8_IMAGE_STATUS Control Register.
	0	**Max Frame bit count Rate Control Enable Mask** Format: U1 Enable Max. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is greater than or equal to FrameBitRateMax. **Value** / **Name** / **Description** 1 / / If (Total Frame Level Bit Counter) >= (Frame Bit Rate Maximum Limit) Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS control register. 0 / / Do not update bit[0] of MFX_VP8_IMAGE_STATUS control register.
3	31:28	**Reserved** Format: MBZ
	27:16	**Max Intra MB Bit Count Limit** Format: U12 12-bit bit count for Max Intra MB Limit.
	15:12	**Reserved** Format: MBZ
	11:0	**Max Inter MB bit count** Format: U12 12-bit bit count for Max Inter MB Limit.
4	31	**Frame Bitrate Min Unit Mode** Format: U1 This field is the Frame Bitrate Minimum Limit Units.

MFX_VP8_Encoder_CFG

Value	Name	Description
0h	Compatibility Mode	Frame BitRate Min Unit is in old mode **(128b/16Kb)**
1h	New Mode	Frame BitRate Min Unit is in new mode **(32byte/4Kb)**

30 **Frame Bit Rate Min Unit**

Format:	U1

This field is Frame Bitrate Minimum Mode.

Value	Name
0	32-B
1	4-KB

29:16 **Frame Bit Rate Min**

Format:	U14

If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values

15 **Frame Bitrate Max Unit Mode**

Format:	U1

This field is the Frame Bitrate Maximum Limit Units.

Value	Name	Description
0h	Compatibility Mode	Frame BitRate Max Unit is in old mode **(128b/16Kb)**
1h	New Mode	Frame BitRate Max Unit is in new mode **(32byte/4Kb)**

14 **Frame Bit Rate Max Unit**

Format:	U1

This field is Frame Bitrate Maximum Mode

Value	Name
0	32-B
1	4-KB

13:0 **Frame Bit Rate Max**

Format:	U14

If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values

MFX_VP8_Encoder_CFG		
5	31:24	**Frame Delta QIndex Max[3]** This field is the Frame level delta Qindex for total bit-count above FrameBitRateMax - First 1/8 Region. This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of (FrameBitRateMax, (FrameBitRateMax + FrameBitRateMaxDelta » 3)].
	23:16	**Frame DeltaQ Index Max[2]** This field is the Frame level delta Qindex for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4. This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3), (FrameBitRateMax+ FrameBitRateMaxDelta » 2)].
	15:8	**Frame Delta QIndex Max[1]** This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/4 and Below 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and 1/2 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of [(FrameBitRateMax+ FrameBitRateMaxDelta » 2), (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].
	7:0	**Frame Delta QIndex Max [0]** This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of **FrameBitRateMax**; i.e., In the range of [(FrameBitRateMax + FrameBitRateMaxDelta » 1), ∞ (Infinite)].
6	31:24	**Frame Delta QIndex Min[3]** This field is the Frame level delta QINDEX for total bit-count below FrameBitRateMin - First 1/8 Region. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 3),

MFX_VP8_Encoder_CFG		
		FrameBitRateMin].
	23:16	**Frame Delta QIndex Min[2]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin- FrameBitRateMinDelta » 3)].
	15:8	**Frame Delta QIndex Min[1]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1), (FrameBitRateMin - FrameBitRateMinDelta » 2)].
	7:0	**Frame Delta QIndex Min[0]** This field is the Frame Level Delta QINDEX for bit-count below FrameBitRateMin - Below 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of **FrameBitRateMin**; i.e., In the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).
7	31:0	**Per Segment Frame Delta QIndex Max[1]** Reserved
8	31:0	**Per Segment Frame Delta QIndex Min[1]** Reserved
9	31:0	**Per Segment Frame Delta QIndex Max[2]** Reserved
10	31:0	**Per Segment Frame Delta QIndex Min[2]** Reserved
11	31:0	**Per Segment Frame Delta QIndex Max[3]** Reserved
12	31:0	**Per Segment Frame Delta QIndex Min[3]** Reserved

MFX_VP8_Encoder_CFG

13	31:24	**Frame Delta Loop Filter Max[3]** Format: U8 This field is the Frame level delta LoopFilter for total bit-count above FrameBitRateMax - First 1/8 region. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta » 3)].
	23:16	**Frame Delta Loop Filter Max[2]** Format: U8 This field is the Frame level delta LoopFilter for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3) and (FrameBitRateMax + FrameBitRateMaxDelta » 2)].
	15:8	**Fram eDelta Loop Filter Max[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above1/ 4 and Below 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and1/2 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 2) and (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].
	7:0	**Frame Delta Loop Filter Max[0]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta. i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 1), infinite).

MFX_VP8_Encoder_CFG

14	31:24	**Frame Delta Loop Filter Min[3]** Format: U8 This field is the Frame level delta LOOPFILTER for total bit-count below FrameBitRateMin - First 1/8 region. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 3), FrameBitRateMin).
	23:16	**Frame Delta Loop Filter Min[2]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin - FrameBitRateMinDelta » 3)).
	15:8	**Frame Delta Loop Filter Min[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1) and (FrameBitRateMin - FrameBitRateMinDelta » 2)).
	7:0	**Frame Delta Loop Filter Min[0]** Format: U8 This field is the Frame Level Delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/ 2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta. i.e., in the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).

MFX_VP8_Encoder_CFG

15	31:0	**Per Segment Frame Delta LoopFilter Max[1]**
16	31:0	**Per Segment Frame Delta LoopFilter Min[1]**
17	31:0	**Per Segment Frame Delta LoopFilter Max[2]**
18	31:0	**Per Segment Frame Delta LoopFilter Min[2]**
19	31:0	**Per Segment Frame Delta LoopFilter Max[3]**
20	31:0	**Per Segment Frame Delta LoopFilter Min[3]**
21	31	**Reserved**
	30:16	**FrameBitRateMinDelta** Format: U15 This field is used to select the frame delta QINDEX when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. **Value** / **Name** / **Description** [0-4095] / / When FrameBitrateMinUnit is in Bytes, this range is in Bytes. When FrameBitrateMinUnit is in KB, this range is in KB units.
	15	**Reserved**
	14:0	**Frame Bit Rate Max Delta** Format: U15 This field is used to select the frame delta QINDEX when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. **Value** / **Name** / **Description** [0-4095] / / When FrameBitrateMinUnit is in Bytes, this range is in Bytes. When FrameBitrateMinUnit is in KB, this range is in KB units.
22	31:24	**Reserved** Format: MBZ
	23	**Show Frame** Format: U1 VP8 Frame Tag, Show Frame Field

MFX_VP8_Encoder_CFG

	22:20	**Bitstream Format Version** Format: U3 VP8 Frame Tag, Verison Field
	19:18	**Reserved** Format: MBZ
	17:16	**Min Frame WSize Unit** Format: U2 Value / Name / Description: 0h — Compatibility Mode — MinFrameWSizeUnit is in old mode (128b/16Kb) 1h — New Mode — MinFrameWSizeUnit is in new mode (32byte/4Kb)
	15:0	**Min Frame WSize** Exists If: //Encoder Only This field (in Word, 16-bit) is specified to compensate for Intel® Rate Control. Zero padding would be performed.
23	31:16	**Vertical_Size_Code** Format: U16 Frame Tag Vertical Size Code, composed of {VerticalScale[15:14], FrameHeight[13:0]}
	15:0	**Horizontal_Size_Code** Format: U16 Frame Tag Horizontal Size Code, composed of {HorizontalScale[15:14], FrameWidth[13:0]}
24	31:0	**Frame Header Bit Count** Format: U32 Binarized Header Bit Count.
25	31:0	**Frame Header Bin Buffer Qindex Update Pointer** Format: U32 Binarized Header Qindex Update Pointer If Segment Enabled and UpdateSegmentFeature enabled, 4 per segment Qindices would be updated in Binarized header (Only ABS mode supported). Else

MFX_VP8_Encoder_CFG

		Base Qindex would be updated
26	31:0	**Frame Header Bin Buffer LoopFilter Update Pointer** Format: U32 Binarized Header LoopFilter Update Pointer If Segment Enabled and UpdateSegmentFeature enabled, 4 per segment LoopFilters would be updated in Binarized header (Only ABS mode supported). Else Base LoopFilter would be updated.
27	31:0	**Frame Header Bin Buffer Token Update Pointer** Format: U32 Binarized Header TokenUpdate Pointer
28	31:0	**Frame Header Bin Buffer MVUpdate Pointer** Format: U32 Binarized Header MVUpdate Pointer.
29 **Programming Notes:** In this project, the only value permitted for CV7 through CV0 is 0xf The only value permitted for CV7 through CV0 is 0xf	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**
	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0** Format: U4 If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped). **For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:** none CV7 CV5 CV4 CV7 CV6 CV4 CV3

MFX_VP8_Encoder_CFG

		CV5 CV4 CV2 CV1 CV4 CV3 CV1 CV0 **For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:** none CV6 CV3 CV1 CV7 CV6 CV3 CV1 CV5 CV4 CV2 CV0 CV5 CV4 CV2 CV0 **Value:** 0-15 **Name:**

MFX_VP8_PAK_OBJECT

MFX_VP8_PAK_OBJECT

Source: VideoCS

Length Bias: 2

The MFX_VP8_PAK_OBJECT command is the second primitive command for the VP8 Encoding Pipeline.
The MV Data portion of the bitstream is loaded as indirect data object.Before issuing a MFX_VP8_PAK_OBJECT command, all VP8 MFX states need to be valid; therefore the commands used to set these states need to have been issued prior to the issue of this command.
MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command.
Internal counter will keep track of the current MB address, starting from the first MB.
MFX_VP8_PAK_OBJECT command follows the MbType definition like MFD.
Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VP8_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 4h VP8_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 5h DWORD_COUNT_n Format: =n Length -2
1	31:30	**Reserved** Format: MBZ

MFX_VP8_PAK_OBJECT

	29	**Enable Inline MV data** Format: Enable This field denotes if the MV data will be sent inline following the other inline data instead of being indirect.
	28:10	**Reserved** Format: MBZ
	9:0	**Indirect PAK-MV Data Length** Format: U10 This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect PAK-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size. Value: [0,512MB) — Name:
3..6	127:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as Inline Data Description - VP8 PAK OBJECT. It has a fixed size of 4 DWs. Its definition is described in the next section.

MFX_VP8_PIC_STATE

MFX_VP8_PIC_STATE

Source: VideoCS

Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_IMG_STATE.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 0h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 0h MFX_VP8_PIC_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Value / Name / Description 000h / Excludes DWord (0,1) **[Default]** / A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 024h / / Used for normal decode and encode mode
1	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ

MFX_VP8_PIC_STATE

	23:16	**Frame Height Minus 1** Exists If: //Decoder / Encoder Format: U8 Picture Height in integer number of MBs minus 1, so the min pic height can be program is 16 rows of pixels.
	15:8	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	7:0	**Frame Width Minus 1** Exists If: //Decoder / Encoder Format: U8 Picture Width in integer number of MBs minus 1, so the min pic width can be program is 16 pixels.
2	31:26	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	25:24	**Log2 Num of Partition** Exists If: //Decoder / Encoder Format: U2 Value / Name: 0 – 1 Token partition 1 – 2 Token partition 2 – 4 Token partition 3 – 8 Token partition
	23:19	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	18:16	**Deblock Sharpness Level** Exists If: //Decoder / Encoder Format: U3 Specifiy the sharpness level, as one of the regular deblocking strength control parameters. **Programming Notes** Set to 0 to disable the use of sharpness control.

MFX_VP8_PIC_STATE

15:14 Reserved

Exists If:	//Decoder / Encoder
Format:	MBZ

13 Alternate Ref Pic MV SignBias Flag

Exists If:	//Decoder / Encoder

Alternate Reference Picture MV sign bias flag, specified for non-key frame only.

12 Golden Ref Picture MV SignBias Flag

Exists If:	//Decoder / Encoder

Golden Reference Picture MV sign bias flag, specified for non-key frame only.

11 Mode Reference Loop Filter Delta Enabled

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Mode or Reference Loop Filter Delta Adjustment for current frame is disabled.
1		Mode or Reference Loop Filter Delta Adjustment for current frame is enabled.

10 MB NoCoeff SkipFlag

Exists If:	//Decoder / Encoder

Frame level control if Skip MB (with no non-zero coefficient) is allowed or not.

Value	Name	Description
0		All MBs will have its MB level signaling mb_skip_coeff forced to 0. That is, no skip of coefficient record in the bitstream (even their values are all 0s)
1		Skip MB is enabled in the per MB record.

9 Update MBSegment Map Flag

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Disable segmentation update
1		Enable segmentation update, and to enable reading segment_id for each MB.

8 Segment Enable Flag

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Disable Segmentation processing in the current frame
1		Enable Segmentation processing in the current frame

MFX_VP8_PIC_STATE

	7	**Segmentation ID StreamIn Enable** Exists If: //Decoder Only Value / Name 0 StreamIn Disabled 1 StreamIn Enabled Programming Notes When 0, no input needed.
	7:6	**Reserved** Exists If: //Encoder Only Format: MBZ
	6	**Segmentation ID StreamOut Enable** Exists If: //Decoder Only Value / Name 0 StreamOut Disabled 1 StreamOut Enabled Programming Notes When 0, no output needed.
	5	**sKeyFrameFlag** Exists If: //Decoder / Encoder Value / Name 0 Non-Key Frame (P-Frame) 1 Key Frame (I-Frame)
	4	**DBLKFilterType** Exists If: //Decoder / Encoder To specify VP8 Profile of operation. Value / Name / Description 0 Use a full feature normal deblocking filter 1 Use a simple filter for deblocking
	3:2	**Reserved** Exists If: //Decoder / Encoder Format: MBZ

MFX_VP8_PIC_STATE

	1	**Chroma Full Pixel MC Filter Mode** Exists If: //Decoder / Encoder To specify VP8 Profile of operation. Value / Name / Description: 0 — Chroma MC filter operates in sub-pixel mode 1 — Chroma MC filter only operates in full pixel position, i.e. no sub-pixel interpolation.
	0	**MC Filter Select** Exists If: //Decoder / Encoder To specify VP8 Profile of operation. Value / Name / Description: 0 — 6-tap filter (regular filter mode) 1 — 2-tap bilinear filter (simple profile/version mode)
3	31:30	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	29:24	**DBLKFilterLevel for Segment3** Exists If: //Decoder / Encoder Format: U6 Value / Name / Description: 0 — Signifies disable in loop deblocking operation — This is used to set a VP8 profile without in loop deblocker. **Programming Notes** There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.
	23:22	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	21:16	**DBLKFilterLevel for Segment2** Exists If: //Decoder / Encoder Format: U6 Value / Name / Description: 0 — Signifies disable in loop deblocking operation — This is used to set a VP8 profile without in loop deblocker.

MFX_VP8_PIC_STATE

		Programming Notes There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.
	15:14	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	13:8	**DBLKFilterLevel for Segment1** Exists If: //Decoder / Encoder Format: U6 **Value** / **Name** / **Description** 0 / Signifies disable in loop deblocking operation / This is used to set a VP8 profile without in loop deblocker. **Programming Notes** There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.
	7:6	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	5:0	**DBLKFilterLevel for Segment0** Exists If: //Decoder / Encoder Format: U6 **Value** / **Name** / **Description** 0 / Signifies disable in loop deblocking operation / This is used to set a VP8 profile without in loop deblocker. **Programming Notes** There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.
4	31	**Reserved** Exists If: //Encoder Only Format: MBZ
	30:24	**Seg 3 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 3

MFX_VP8_PIC_STATE

	23	**Reserved** Exists If: //Encoder Only Format: MBZ
	22:16	**Seg 2 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 2
	15	**Reserved** Exists If: //Encoder Only Format: MBZ
	14:8	**Seg 1 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 1
	7	**Reserved** Exists If: //Encoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Decoder Only Format: MBZ
	6:0	**Seg 0 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 0. **Programming Notes** This is the **[Default]** Qindex
5	31:29	**Reserved** Exists If: //Encoder Only Format: MBZ
	28	**UVac Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for UVac

MFX_VP8_PIC_STATE

	27:24	**UVac QindexDelta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for UVac
	23:21	**Reserved** Exists If: //Encoder Only Format: MBZ
	20	**UVdc Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for UVdc
	19:16	**UVdc Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for UVdc
	15:13	**Reserved** Exists If: //Encoder Only Format: MBZ
	12	**Y2ac Qindex Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2ac
	11:8	**Y2ac Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2ac
	7:5	**Reserved** Exists If: //Encoder Only Format: MBZ

MFX_VP8_PIC_STATE

	4	**Y2ac Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2dc This is the **[Default]** Qindex Delta Sign
	31:0	**Reserved** Exists If: //Decoder Only Format: MBZ
	3:0	**Y2dc Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2dc This is the **[Default]** Qindex Delta
6	31:5	**Reserved** Exists If: //Encoder Only Format: MBZ
	4	**Y1dc Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y1dc This is the **[Default]**Qindex Delta Sign
	31:0	**Reserved** Exists If: //Decoder Only Format: MBZ
	3:0	**Y1dc Qindex Delta** Exists If: //Encoder Only Absolute Quantization index delta for Y1dc This is the **[Default]**Qindex Delta
7	31:15	**Reserved** Exists If: //Encoder Only Format: MBZ

MFX_VP8_PIC_STATE

	14:8	**Clamp Qindex high** Exists If: //Encoder Only Format: U7 Maximum Clamp Value for Qindex used in quantization.
	7	**Reserved** Exists If: //Encoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Decoder Only Format: MBZ
	6:0	**Clamp Qindex Low** Exists If: //Encoder Only Format: U7 Minimum Clamp Value for Qindex used in quantization.
8	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [1][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [1][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
9	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ

MFX_VP8_PIC_STATE

	24:16	**Quantizer Value [1][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [1][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
10	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [2][BlockType1=Y1AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [2][BlockType0=Y1DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
11	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ

MFX_VP8_PIC_STATE

	24:16	**Quantizer Value [2][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [2][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
12	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [2][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [2][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
13	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ

MFX_VP8_PIC_STATE

	24:16	**Quantizer Value [3][BlockType1=Y1AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType0=Y1DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
14	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [3][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
15	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ

MFX_VP8_PIC_STATE

	24:16	**Quantizer Value [3][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
16	31:6	**CoeffProbability StreamIn Base Address** Exists If: //Decoder Only Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format. (not tile for better performance) It is specified for non-key frame only. It is the final computed probability table for parsing Coeff in the bitstream. The buffer is unsigned 8-bit * 1056 entries (CoeffProbs[4][8][3][11].
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	5:0	**Reserved** Exists If: //Decoder Only Format: MBZ
17	31:16	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ

MFX_VP8_PIC_STATE

	15:0	**CoeffProbability StreamIn Address** Exists If: //Decoder Only This field is for the upper range of CoeffProbability StreamIn Address
18	31:15	**Reserved** Exists If: //Decoder Only Format: MBZ
	14:13	**CoeffProbability StreamIn - Tiled Resource Mode** Exists If: //Decoder Only Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12:11	**Reserved** Exists If: //Decoder Only Format: MBZ
	10	**CoeffProbability StreamIn - Memory Compression Mode** Exists If: //Decoder Only Format: U1 Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

	9	**CoeffProbability StreamIn - Memory Compression Enable** Exists If: //Decoder Only Format: Enable Memory compression will be attempted for this surface.
	31:7	**Reserved** Exists If: //Encoder Only Format: MBZ

MFX_VP8_PIC_STATE

	8:7	**CoeffProbability StreamIn - Arbitration Priority Control** Exists If: //Decoder Only Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:1	**CoeffProbability StreamIn Address - Index to Memory Object Control State (MOCS) Tables** Exists If: //Encoder Only Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
19	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MBSegmentIDTreeProbs[2]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
	15:8	**MBSegmentIDTreeProbs[1]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
	7:0	**MBSegmentIDTreeProbs[0]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.

MFX_VP8_PIC_STATE

20	31:24	**MBNoCoeffSkipFalseProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the MBNoCoeffSkip Flag in the bistream.
	23:16	**IntraMBProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the intra or inter MB type flag in the bitstream.
	15:8	**InterPredFromLastRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
	7:0	**InterPredFromGRefRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
21	31:24	**YModeProb[3]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	23:16	**YModeProb[2]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	15:8	**YModeProb[1]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.

MFX_VP8_PIC_STATE

	7:0	**YModeProb[0]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
22	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**UVModeProb[2]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	15:8	**UVModeProb[1]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	7:0	**UVModeProb[0]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
23	31:24	**MVUpdateProbs[0][3]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][2]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	15:8	**MVUpdateProbs[0][1]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][0]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
24	31:24	**MVUpdateProbs[0][7]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][6]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][5]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][4]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

25	31:24	**MVUpdateProbs[0][11]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	23:16	**MVUpdateProbs[0][10]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	15:8	**MVUpdateProbs[0][9]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	7:0	**MVUpdateProbs[0][8]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
26	31:24	**MVUpdateProbs[0][15]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][14]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][13]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	7:0	**MVUpdateProbs[0][12]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
27	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MVUpdateProbs[0][18]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][17]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][16]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
28	31:24	**MVUpdateProbs[1][3]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][2]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	15:8	**MVUpdateProbs[1][1]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][0]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
29	31:24	**MVUpdateProbs[1][7]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][6]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][5]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][4]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

30	31:24	**MVUpdateProbs[1][11]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][10]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][9]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][8]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
31	31:24	**MVUpdateProbs[1][15]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][14]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	15:8	**MVUpdateProbs[1][13]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][12]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
32	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MVUpdateProbs[1][18]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][17]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][16]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
33	31	**Reserved** Exists If: //Decoder / Encoder Format: MBZ

MFX_VP8_PIC_STATE

	30:24	**RefLFDelta3 (for ALTREF FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	23	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	22:16	**RefLFDelta2 (for GOLDEN FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	15	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	14:8	**RefLFDelta1 (for LAST FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.

MFX_VP8_PIC_STATE

	7	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	6:0	**RefLFDelta0 (for INTRA FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
34	31	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	30:24	**ModeLFDelta3 (for SPLITMV mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	23	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	22:16	**ModeLFDelta2 (for Nearest, Near and New mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes**

MFX_VP8_PIC_STATE

		Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	15	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	14:8	**ModeLFDelta1(for ZEROMV mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	7	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	6:0	**ModeLFDelta0 (for B_PRED mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
35	31:0	**Segmentation ID Stream Base Address** Exists If: //Decoder Only Format: StreamAddress[31:0] 64 bytes linear aligned buffer It is specified when **SegmentationIDStreamInEnable** or **SegmentationIDStreamOutEnable** is specified. **Programming Notes** Each cache has only 8 bits for 4 segmentation ID from 4 continuous MBs.

MFX_VP8_PIC_STATE

36 | 31:16 | Reserved

Exists If:	//Decoder Only
Format:	MBZ

15:0 | Segmentation ID Stream Base Address [47:32]

Exists If:	//Decoder Only

This field is for the upper range of **Segmentation ID Stream Base Address**

37 | 31:15 | Reserved

Exists If:	//Decoder Only
Format:	MBZ

14:13 | Segmentation ID Stream - Tiled Resource Mode

Exists If:	//Decoder Only
Format:	U2

For Media Surfaces:

This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12:11 | Reserved

Exists If:	//Decoder Only
Format:	MBZ

10 | Segmentation ID Stream - Memory Compression Mode

Format:	U1

Distinguishes Vertical from Horizontal compression. Please refer to vol1a **Memory Data Formats** chapter, **Media Memory Compression** section for more details.

Value	Name
0	Horizontal Compression Mode
1	Vertical Compression Mode

9 | Segmentation ID Stream - Memory Compression Enable

Format:	Enable

Memory compression will be attempted for this surface.

MFX_VP8_PIC_STATE

	8:7	**Segmentation ID Stream - Arbitration Priority Control** Exists If: //Decoder Only Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:1	**CoeffProbability StreamIn Address - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**

MFX_WAIT

MFX_WAIT

Source: VideoCS

Length Bias: 1

This command can be considered the same as an MI_NOOP except that the command parser will not parse the next command until the following happens

- **AVC or VC1 BSD mode:** The command will stall the parser until completion of the BSD object
- **IT, encoder, and MPEG2 BSD mode:** The command will stall the parser until the object package is sent down the pipelineThis command should be used to ensure the preemption enable window occurs during the time the object command is being executed down the pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Command Subtype** Default Value: 01h MFX_SINGLE_DW Format: OpCode
	26:16	**Sub-Opcode** Default Value: 0h MFX_WAIT Format: OpCode
	15:10	**Reserved** Format: MBZ
	9	**Reserved**
	8	**MFX Sync Control Flag** Description: If set, VCS will stall the parser until all prior MFX objects are completed down the MFX pipeline
	7:6	**Reserved** Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_ARB_CHECK

MI_ARB_CHECK

Source: VideoEnhancementCS

Length Bias: 1

Description

The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.

Programming Notes

This instruction cannot be placed in a batch buffer.

If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.

DWord	Bit	Description
0	31:29	**MI Instruction Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Instruction Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Source: BlitterCS

Length Bias: 1

Description

The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.

Programming Notes

This instruction cannot be placed in a batch buffer.

If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Source: RenderCS

Length Bias: 1

Description

The MI_ARB_CHECK instruction is used to check the ring buffer double buffered head pointer (register UHPTR). This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the updated head pointer register needs to be set for the command streamer to be pre-empted.

Programming Notes

Ring Buffer mode of scheduling:

- The current head pointer is loaded with the updated head pointer register independent of the location of the updated head.
- If the current head pointer and the updated head pointer register are equal, hardware will automatically reset the valid bit corresponding to the UHPTR.
- For pre-emption, the wrap count in the ring buffer head register is no longer maintained by hardware. The hardware updates the wrap count to the value in the UHPTR register.

Execlist mode of scheduling:
MI_ARB_CHK will be used to indicate a command boundary on which Preemption will be honored by Command Streamer in the execlist mode of operation. UHPTR is ignored when processing MI_ARB_CHK in execlist mode.

This instruction can be in either a ring buffer or batch buffer.

MI_ARB_CHK command must not be programmed in INDIRECT_CTX and BB_PER_CTX_PTR buffers.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Source: VideoCS

Length Bias: 1

Description

The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.

Programming Notes

This instruction cannot be placed in a batch buffer.

If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.

DWord	Bit	Description
0	31:29	**MI Instruction Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Instruction Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_ON_OFF

MI_ARB_ON_OFF

Source: CommandStreamer

Length Bias: 1

Description

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command.

This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.) This command should always be used as an off-on pair with the sequence of instructions to be protected from context switch between MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

Execution List Mode of Scheduling: The MI_ARB_ON_OFF instruction is used to disable/enable context switching. Context swiching could be either due to preemption or un-succesfull wait for events or semaphore waits. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.)
Ring Buffer Mode of Scheduling: The MI_ARB_ON_OFF instruction is used to disable preemption on the preemptable commands. SW can explicitly make section of commands in a command buffer non-preemptable by sandwitching them between ARB_OFF and ARB_ON, HW will ingore preemption request (UHPTR Valid) until arbitration is enabled.

Programming Notes

This command must be always be programmed in pairs of off/on in the same command dispatch. Sequence of instructions to be protected from cntext switch or preemption must be programmed between the MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 08h MI_ARB_ON_OFF Format: OpCode

MI_ARB_ON_OFF

	22:2	**Reserved** Format: MBZ
	1	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	0	**Arbitration Enable** Format: Enable This field enables or disables context switches due to pre-emption (a new execlist). Value / Name 1 – Arbitration Enabled **[Default]** 0 – Arbitration Disabled

MI_ATOMIC

MI_ATOMIC

Source: BSpec

Length Bias: 2

Description

MI_ATOMIC is used to carry atomic operation on data in graphics memory. Atomic operations are supported on data granularity of 4B, 8B and 16B. The atomic operation leads to a read-modify-write operation on the data in graphics memory with the option of returning value. The data in graphics memory is modified by doing arithmetic and logical operation with the inline/indirect data provided with the MI_ATOMIC command. Inline/Indirect provided in the command can be one or two operands based on the atomic operation. Ex: Atomic-Compare operation needs two operands while Atomic-Add operation needs single operand and Atomic-increment requires no operand. Refer "Atomics" sub-section under "L3 Cache and URB" section for detailed atomic operations supported. Atomic operations can be enabled to return value by setting "Return Data Control" field in the command, return data is stored to CS_GPR registers.
CS_GPR4/5 registers are updated with memory Return Data based on the "Data Size". Each GPR register is qword in size and occupies two MMIO registers.
Note: Any references to CS_GPR registers in the command should be understood as the CS_GPR registers belonging to the corresponding engines *CS_GPR registers.

Engine Name	Corresponding GPR Registers
RCS	CS_GPR
BCS	BCS_GPR
VCS	VCS_GPR
VECS	VECS_GPR

Indirect Source Operands:
Operand1 is sourced from [CS_GPR1, CS_GPR0]
Operand2 is sourced from [CS_GPR3, CS_GPR2]
Read return Data is stored in [CS_GPR_5, CS_GPR4]

When "Data Size" is DWORD lower dword of CS_GPR4 (Qword) is updated with the dword data returned from memory. When "Data Size" is QWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory.

Programming Notes

- When Inline Data mode is not set, Dwords 3..10 must not be included as part of the command. Dword Length field in the header must be programmed accordingly.
- When Inline Data Mode is set, Dwords3..10 must be included based on the Data Size field of the header. Both Operand-1 and Operand-2 dwords must be programmed based on the Data Size field. Operand-2 must be programmed to 0x0 if the atomic operation doesn't require it. Dword Length field in the header must be programmed accordingly.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td>Command Type
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td>MI Command Opcode
<table>
<tr><td>Default Value:</td><td>2Fh MI_ATOMIC</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td>Memory Type
This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr>
<tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td>Reserved
<table>
<tr><td>Source:</td><td>BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td>Post-Sync Operation
<table>
<tr><td>Source:</td><td>RenderCS</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No Post Sync Operation</td><td>Command is executed as usual.</td></tr>
<tr><td>1h</td><td>Post Sync Operation</td><td>MI_ATOMIC command is executed as a pipelined PIPE_CONTROL flush command with Atomics operation as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding flushes issued prior to this command.
When this bit set following ristiriciton apply to atomic operation:
• Non-Compare atomic operations are supported on data granularity of 4B and 8B. DW3 is the lower dword of the operand and DW4 is the upper dword of the operand for the atomic operation.
• Compare atomic operations are supported on data granularity of 4B. DW3 is Operand-0 and DW4 is Operand-1 for the atomic operation.</td></tr>
</table></td></tr>
</table>

MI_ATOMIC

		• Atomic operations to GGTT/PPGTT memory surface are supported. • Only Inline data mode for atomic operand is supported, no support for indirect data mode. • No support for Return Data Control functionality. • No support for atomic operations on data granularity of 16B. • No support for compare atomic operations on data granularity of 8B.

Programming Notes
Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command.
AWhen this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued.
AWhen this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.

20:19 **Data Size**

This field indicates the size of the operand in dword/qword/octword on which atomic operation will be performed. Data size must match with the Atomic Opcode. Operation Data size could be 4B, 8B or 16B

Value	Name	Description
0h	DWORD	Operand size used by Atomic Operation is DWORD.
1h	QWORD	Operand Size used by Atomic Operation is QWORD.
2h	OCTWORD	Operand Size used by Atomic Operation is OCTWORD.
3h	RESERVED	

18 **Inline Data**

This bit when set indicates the source operands are provided in line within the command. When reset the source operands are in CS_GPR registers.

Programming Notes
CS_GPR registers must be programmed with appropriate values before issuing MI_ATOMIC command with this field reset.

MI_ATOMIC

17 CS STALL

This bit when set command stream waits for completion of this command before executing the next command.

Programming Notes	Source
Render Command Streamer Only: CS will not guarantee atomic operation to be complete upon setting this bit along with Post Sync Operation set. When Post Sync Operation is set, this bit has no significance.	RenderCS

Workaround
Workaround : When CS STALL bit is set, Return Data Control must also be set in MI_ATOMIC command.

16 Return Data Control

Source:	RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS

Description
When Return Data Control is set the read return feature will be enabled during the atomic operation. Data is stored in CS_GPR5/4 registers unconditionally on completion of the atomic operation. On data return CS_GPR5/4 Registers are updated based on the "Data Size" field. When "Data Size" is DWORD lower dword of CS_GPR4 (Qword) is updated with the dword data returned from memory. When "Data Size" is QWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory.

Workaround
Workaround: When Return Data Control bit is set, CS STALL must also be set in MI_ATOMIC command.

15:8 ATOMIC OPCODE

This field selects the kind of atomic operation to be performed. Refer "Atomics" sub-section of "L3 Cache and URB" section of the B-spec for atomic opcode encoding and operation.

7:0 DWord Length

Format:	=n

Total Length - 2. Excludes DWord (0,1).

Value	Name	Exists If
1h	**[Default]**	([Inline Data]==0)
9h		([Inline Data]==1)

MI_ATOMIC

1	31:2	**Memory Address** Format: GraphicsAddress[31:2] This field contains the graphics memory address of the data on which atomic operation has to be performed. Atomic operation can be performed on data granularity of 4B, 8B or 16B and hence the Address has to be correspondingly aligned to 4B,8B or 16B respectively.
	1:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Memory Address High** This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
3	31:0	**Operand1 Data Dword 0** Format: U32 Dword0 of Operand1 when Inline Data mode is set.
4	31:0	**Operand2 Data Dword 0** Format: U32 Dword0 of Operand2 when Inline Data mode is set.
5	31:0	**Operand1 Data Dword 1** Format: U32 Dword1 of Operand1 when Inline Data mode is set.
6	31:0	**Operand2 Data Dword 1** Format: U32 Dword1 of Operand2 when Inline Data mode is set.
7	31:0	**Operand1 Data Dword 2** Format: U32 Dword2 of Operand1 when Inline Data mode is set.
8	31:0	**Operand2 Data Dword 2** Format: U32 Dword2 of Operand2 when Inline Data mode is set.
9	31:0	**Operand1 Data Dword 3** Format: U32 Dword3 of Operand1 when Inline Data mode is set.
10	31:0	**Operand2 Data Dword 3** Format: U32 Dword3 of Operand2 when Inline Data mode is set.

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Source: VideoEnhancementCS Length Bias: 1		
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_BATCH+_BUFFER_END Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Source: BlitterCS Length Bias: 1		
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Source: RenderCS Length Bias: 1		
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Source: VideoCS Length Bias: 1		
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_BATCH+_BUFFER_END Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Source: CommandStreamer

Length Bias: 2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a batch buffer. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*. The batch buffer can be specified as privileged or non-privileged, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes

- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.
- It is essential that the address location beyond the current page be populated inside the GTT. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**Second Level Batch Buffer** The command streamer contains three storage elements; one for the ring head address, one for the batch head address, and one for the second level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the first level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the first level batch buffer address. This allows hardware to mimic a simple 3-level stack. Value / Name / Description 0h / First level batch / Place the batch buffer address in the 1st (traditional) level batch address storage element. 1h / Second / Place the batch buffer address in the

MI_BATCH_BUFFER_START

	level batch	second-level batch address storage element.

Programming Notes
Within a second level batch buffer there can't be any chained batch buffers. MI_BATCH_BUFFER_START command is not allowed inside a second level batch buffer.
Setting this bit when the command is parsed in the ring, will cause the command to be ignored. This bit must only be set in a first level batch buffer.

21:20 **Reserved**

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

21:20 **Reserved**

Source:	RenderCS
Format:	MBZ

19 **Reserved**

Source:	RenderCS
Format:	MBZ

18:17 **Reserved**

Format:	MBZ

16 **Reserved**

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

16 **Add Offset Enable**

Source:	RenderCS
Format:	Enable

If this bit is set then the value stored in the BB_OFFSET MMIO register will be added to the Batch Buffer Start Address and the summation will be used as the address to fetch from memory.
Specific to the render command stream only.

15 **Reserved**

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

MI_BATCH_BUFFER_START

	15	**Predication Enable** Source: RenderCS This bit is used to enable predication of this command. If this bit is set and Bit 0 of the MI_PREDICATE_RESULT_1 register is clear, this command is ignored. Otherwise the command is performed normally. **Specific to the Render command stream only.**
	14:11	**Reserved** Format: MBZ
	10	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	10	**Resource Streamer Enable** Source: RenderCS Format: Enable When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer. **Specific to the Render command stream only.**
	9	**Reserved** Source: RenderCS, BlitterCS Format: MBZ
	9	**Reserved**
	8	**Address Space Indicator** Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain. • Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT. • Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.

MI_BATCH_BUFFER_START

		Value / Name / Description 0h / GGTT / This batch buffer is located in GGTT memory and is privileged. 1h / PPGTT / This batch buffer is located in PPGTT memory and is Non-Privileged. **Programming Notes** This field must be '0' unless the Per-Process GTT Enable is '1'
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total - Bias. Excludes DWord (0,1).
1..2 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example, GraphicsAddress[47:12] is a 4KB page address.	63:48	**Reserved** Format: MBZ These bits are reserved for future expansion of the supported Virtual Address range, and are not to be used for other purposes.
	47:2	**Batch Buffer Start Address** Format: GraphicsAddress[47:2]
	1:0	**Reserved** Format: MBZ

MI_CLFLUSH

MI_CLFLUSH

Source: RenderCS

Length Bias: 2

Flushes out the page given in the command out to system memory. This command is specific to the render engine and is not privileged.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 27h Store DW MI_CLFLUSH Format: OpCode
	22	**Use Global GTT** 0h – Per Process Graphics Address 1h – Global Graphics Address: This command will use the global GTT to translate the Address.
	21:10	**Reserved** Format: MBZ
	9:0	**DWord Length** Format: n Total Length - 2 1h – **[Default]** – Excludes DWord (0,1)
1..2	63:12	**Page Base Address** Format: GraphicsAddress63-12 4KB aligned Page Address for which software requires a hardware flush to DRAM. GraphicsAddress is 64-bit value [63:0], but only a portion of it is used by the hardware. The uppermost bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] is a 4KB page address.
	11:6	**Starting Cacheline Offset** Format: U6 Zero based starting cacheline offset in to the Page Base Address.

<table>
<tr><th colspan="3">MI_CLFLUSH</th></tr>
<tr><td></td><td>5:0</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>3..n</td><td>31:0</td><td>DW Representing a Half Cache Line<table><tr><td>Format:</td><td>MBZ</td></tr></table>The information given to hardware is the DW itself, not the contents. Hardware uses the DW count of the command to determine the offset from the base to flush out. The offset is ½ cache line (8 DW = 1HW) granular so for a full page, the command will need 4096 bytes / 4 bytes per DW / 8 DW per HW = 128 DW.<table><tr><th>Programming Notes</th></tr><tr><td>Always even number of "DW Representing 1/2 cacheline" terms must be programmed.</td></tr></table></td></tr>
</table>

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Source: CommandStreamer

Length Bias: 2

Description

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command. Termination of second level batch buffer due to this command will also terminate the parent/first level batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Default Value: 0h Format: Boolean If set, this command will use the global GTT to translate the **Compare Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Compare Address**.
	21	**Compare Semaphore** Default Value: 0h Format: Boolean If set, the value from the Compare Data Dword is compared to the value from the Compare Address in memory. If the value at Compare Address is greater than the Compare Data Dword, execution of current command buffer should continue. If clear, the parser will continue to the next command and not exit the batch buffer.
	20	**Reserved**

MI_CONDITIONAL_BATCH_BUFFER_END

	19	**Compare Mask Mode** Value / Name / Description: 0h / Compare Mask Mode Disabled / Compare address points to Dword in memory consisting of Data Dword(DW0). HW will compare Data Dword(DW0) against Semaphore Data Dword. 1h / Compare Mask Mode Enabled / Compare address points to Qword in memory consisting of compare Mask (DW0) and Data Dword(DW1). HW will do AND operation on Mask(DW0) with Data Dword(DW1) and then compare the result against Semaphore Data Dword. **Programming Notes** When "Compare Mask Mode" is enabled, "Compare Address" must be qword aligned.
	18	**Reserved**
	17:15	**Reserved**
	14:12	**Reserved**
	11:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at Compare Address is greater than this dword, the execution of the command buffer should continue.
2..3 Qword address to fetch Data Qword from memory. This field specifies the 4GB aligned base address of Gfx 4GB	63:48	**Reserved** Format: MBZ
	47:3	**Compare Address** Format: GraphicsAddress[47:3]

MI_CONDITIONAL_BATCH_BUFFER_END

virtual address space within the host's 64-bit virtual address space. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. Upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] is a 4KB page address.	2:0	**Reserved** Format: MBZ

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Source: VideoCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non-privileged" batch buffer to access global virtual space; doing so will be treated as privilege access violation. Refer to the "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to learn more about HW behavior on encountering a privilege access violation.
This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address 1h – Global Graphics Address – This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address

MI_COPY_MEM_MEM

		1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	20:8	**Reserved**		
		Format:		MBZ
	7:0	**DWord Length**		
		Default Value:		3 Excludes DWord (0,1)
		Format:		=n Total Length - 2
1..2	63:2	**Destination Memory Address**		
		Format:		GraphicsAddress63-2
		Surface Type: MMIO Register		
	1:0	**Reserved**		
		Format:		MBZ
3..4	63:2	**Source Memory Address**		
		Format:		GraphicsAddress63-2
		Surface Type: MMIO Register		
	1:0	**Reserved**		
		Format:		MBZ

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Source: VideoEnhancementCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non-privileged" batch buffer to access global virtual space; doing so will be treated as privilege access violation. Refer to the "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to learn more about HW behavior on encountering a privilege access violation.

This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description 0h / Per Process Graphics Address / 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description 0h / Per Process Graphics Address /

MI_COPY_MEM_MEM

		1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	20:8	**Reserved**		
		Format:		MBZ
	7:0	**DWord Length**		
		Default Value:		3 Excludes DWord (0,1)
		Format:		=n Total Length - 2
1..2	63:2	**Destination Memory Address**		
		Format:		GraphicsAddress63-2
		Surface Type: MMIO Register		
	1:0	**Reserved**		
		Format:		MBZ
3..4	63:2	**Source Memory Address**		
		Format:		GraphicsAddress63-2
		Surface Type: MMIO Register		
	1:0	**Reserved**		
		Format:		MBZ

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Source: BlitterCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_COPY_MEM_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description 0h / Per Process Graphics Address / 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_COPY_MEM_MEM

	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. **Value** / **Name** / **Description** 0h / Per Process Graphics Address / 1h / Global Graphics Address / This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3 Excludes DWord (0,1) Format: =n Total Length - 2
1..2 Surface Type: MMIO Register GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.	63:48	**Reserved** Format: MBZ
	47:0	**Destination Memory Address** Format: GraphicsAddress[47:0]
3..4 Surface Type: MMIO Register GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.	63:48	**Reserved** Format: MBZ
	47:0	**Source Memory Address** Format: GraphicsAddress[47:0]

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Source: RenderCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_COPY_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value 0h: Per Process Graphics Address Value 1h: Global Graphics Address — It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.
	21	**Use Global GTT Destination** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. This bit will determine write to memory uses Global or Per Process GTT. Value 0h: Per Process Graphics Address

MI_COPY_MEM_MEM

		1h: Global Graphics Address — This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3 Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:2	**Destination Memory Address** Format: GraphicsAddress[63:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Format: MBZ
3..4	63:2	**Source Memory Address** Format: GraphicsAddress[63:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Format: MBZ

MI_DISPLAY_FLIP

MI_DISPLAY_FLIP

Source:	RenderCS, BlitterCS
Length Bias:	2

The MI_DISPLAY_FLIP command is used to request a specific display plane to switch (flip) to display a new buffer. The buffer is specified with a starting address and pitch. The tiled attribute of the buffer start address is programmed as part of the packet.

The operation this command performs is also known as a "display flip request" operation - in that the flip operation itself will occur at some point in the future. This command specifies when the flip operation is to occur: either synchronously with vertical retrace to avoid tearing artifacts (possibly on a future frame), or asynchronously (as soon as possible) to minimize rendering stalls at the cost of tearing artifacts.

Programming Notes

1. This command simply requests a display flip operation. Command execution then continues normally. There is no guarantee that the flip (even if asynchronous) will occur prior to subsequent commands being executed. (Note that completion of the MI_FLUSH command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command can be used to provide this synchronization - by pausing command execution until a pending flip has actually completed. This synchronization can also be performed by use of the Display Flip Pending hardware status. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.
2. After a display flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or rendering operations. For multi-buffering (e.g., double buffering) operations, this will typically require updating SURFACE_STATE or the binding table to change the rendering (back) buffer. In addition, prior to any subsequent clear or rendering operations, software must typically ensure that the new rendering buffer is not actively being displayed. Again, the MI_WAIT_FOR_EVENT command or Display Flip Pending hardware status can be used to provide this synchronization. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.
3. The display buffer command uses the X and Y offset for the tiled buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For tiled buffers, the display subsystem uses the X and Y offset in generation of the final request to memory. The offset is always updated on the next vblank for both Synchronous and Asynch Flips. It is not necessary to have a flip enqueued to update the X and Y offset.
4. The display buffer command uses the linear dword offset for the linear buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For linear buffers, the display subsystem uses the dword offset in generation of the final request to memory.
    - For synchronous flips the offset is updated on the next vblank. It is not necessary to have a sync flip enqueued to update the dword offset.
5. DWord 3 (Left Eye Display Buffer Base Address) must not be set with synchronous flips or asynchronous flips. It is only allowed to be sent with stereo 3D flips.

MI_DISPLAY_FLIP

Asynchronous flip completion time depends greatly on how much data has been prefetched for power savings, and can take up to 1 full frame to complete. For faster flip completion, disable FBC and render compression and allocate a small amount of data buffer for the plane.

"Command Streamer Plane Number" mapping to "Display Plane Name" are listed in dsiplay B-spec -"Plane capability and Interoperability".

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 14h MI_DISPLAY_FLIP Format: OpCode
	22	**Async Flip Indicator** Format: Enable This bit should always be set if DW2 [1:0] == '01' (async flip). This field is required due to HW limitations. This bit is used by the render pipe while DW2 is used by the display hardware.
	21:19	**Reserved** Format: MBZ
	18:17	**Reserved**
	16:14	**Reserved** Format: MBZ
	13	**Reserved** Format: MBZ
	12:8	**Display Plane Select** Value / Name: 0h Display Plane 1 1h Display Plane 2 2h Display Plane 3 3h Reserved 4h Display Plane 4 5h Display Plane 5 6h Display Plane 6 7h Display Plane 7 8h Display Plane 8 9h Display Plane 9

MI_DISPLAY_FLIP

		Ah — Display Plane 10
		Bh — Display Plane 11
		Ch — Display Plane 12
		[Dh-1Fh] — Reserved
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). For Synchronous Flips and Asynchronous Flips, this field must be programmed to 1h for a total length of 3. For Stereo 3D Flips, this field must be programmed to 2h for a total length of 4. 1h — **[Default]** — ([Flip Type]!='Stereo 3D Flip') 2h — **[Default]** — ([Flip Type]=='Stereo 3D Flip')
1	31	**Stereoscopic 3D Mode** Default Value: 0h Format: Enable This bit must be set if the Extra Display Buffer Address is part of this command. This bit is used to notify the display there is an extra DW before processing the Display Flip.
	30:16	**Reserved** Format: MBZ
	15:6	**Display Buffer Pitch** Default Value: 0h Format: U10 *For Synchronous Flips and Stereo 3D Flips only,* this field specifies the pitch of the new display buffer. For Asynchronous Flips, this parameter is programmed so that all the flips in a flip chain should maintain the same pitch as programmed with the last synchronous flip or stereo 3D flip or direct through MMIO. See the Display Plane Stride register for details.
	5:3	**Reserved** Format: MBZ

MI_DISPLAY_FLIP

	2:0	**Tile Parameter** Format: Enable For Asynchronous Flips, this parameter cannot be changed. All the flips in a flip chain should maintain the same tile parameter as programmed with the last synchronous flip or direct through MMIO. **Value / Name / Description** 0h / Linear **[Default]** / For Syncronous Flips Only 1h / Tiled X 2h-3h / Reserved 4h / Tiled Y Legacy (Y B) 5h / Tiled Y F
2	31:12	**Display Buffer Base Address** Format: GraphicsAddress[31:12] This field specifies Bits 31:12 of the Graphics Address of the new display buffer. In stereo 3D mode this is the right eye base address. In non-stereo 3D mode this is the only base address. (Refer to the Display Address Start Address Register description in the Display Registers chapter). **Programming Notes** • The Display buffer must reside completely in Main Memory. • This address is always translated via the global (rather than per-process) GTT.
	11:3	**Reserved** Format: MBZ
	2	**Reserved**
	1:0	**Flip Type** This field specifies whether the flip operation should be performed asynchronously to vertical retrace. **Value / Name / Description** 00b / Sync Flip **[Default]** / The flip will occur during the vertical blanking interval - thus avoiding any tearing artifacts. 01b / Async Flip / The flip will occur "as soon as possible" - and may exhibit tearing artifacts 10b / Stereo 3D Flip / The flip will occur during the vertical blanking interval (left or right eye blank selectable through display MMIO register) - thus avoiding any tearing artifacts. 11b / Reserved / **Programming Notes** • The Display Buffer Pitch and Tile parameter cannot be changed for asynchronous flips

MI_DISPLAY_FLIP

		(i.e., the new buffer must have the same pitch/tile format as the previous buffer). • For Asynch Flips the Buffers used must be 32KB aligned. • Asynch flips are supported on primary or universal planes only. • For Stereo 3D flips, both left and right eye buffers must have the same pitch and tile format.
3	31:12	**Left Eye Display Buffer Base Address** Format: GraphicsAddress[31:12] This field specifies Bits 31:12 of the Graphics Address of the new display buffer for the stereo 3D left eye. In non-stereo 3D mode this address is not used. (Refer to the Display Address Start Address Register description in the Display Registers chapter). **Programming Notes** • The Display buffer must reside completely in Main Memory. • This address is always translated via the global (rather than per-process) GTT.
	11:0	**Reserved** Format: MBZ

MI_FLUSH_DW

MI_FLUSH_DW

Source: VideoEnhancementCS

Length Bias: 2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:

- Flush any dirty data to memory.
- Invalidate the TLB cache inside the hardware

Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved**
	21	**Store Data Index** Format: U1 **Description** **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved** Format: MBZ
	18	**TLB Invalidate** Format: U1 **Description** If ENABLED, all TLBs belonging to Video Enhancement Engine will be invalidated once the flush

MI_FLUSH_DW

operation is complete.
This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.

If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.

17 **Reserved**

Format:	MBZ

16 **Reserved**

Format:	MBZ

15:14 **Post-Sync Operation**

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address
2h	Reserved	Reserved
3h	Write TIMESTAMP register	Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.

Programming Notes

If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

13:10 **Reserved**

Format:	MBZ

9 **Flush LLC**

Format:	Enable

If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.

8 **Notify Enable**

Format:	U1

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

7 **Reserved**

Format:	MBZ

6 **Reserved**

Format:	MBZ

MI_FLUSH_DW

	5:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** 3h: Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]**
1	31:3	**Address** Format: GraphicsAddress[31:3]U28 This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.
	2	**Destination Address Type** Defines address space of Destination Address **Value** / **Name** / **Description** 0h / PPGTT / Use PPGTT address space for DW write 1h / GGTT / Use GGTT address space for DW write **Programming Notes** Ignored if "No write" is the selected in Operation.
	1:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Address High** Format: GraphicsAddress[47:32]U64 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space
3..4	31:0	**Immediate Data** This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FLUSH_DW

MI_FLUSH_DW

Source: BlitterCS

Length Bias: 2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to: Flush any dirty data to memory. Invalidate the TLB cache inside the hardware

Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved** Format: U1
	21	**Store Data Index** Format: U1 **Description** **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved** Format: MBZ
	18	**TLB Invalidate** Format: U1 **Description** If ENABLED, all TLBs belonging to Blitter Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.

MI_FLUSH_DW

If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.

17 **Reserved**

Format:	MBZ

16 **Reserved**

Format:	MBZ

15:14 **Post-Sync Operation**

BitFieldDesc

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data QWord	Write the QWord containing Immediate Data Low, High DWs to the Destination Address
2h	Reserved	Reserved
3h	Write TIMESTAMP Register	Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.

Programming Notes

If executed in a PPGTT (non-secure) batch buffer, the post-sync op is always inhibited. This command does not write anything out to memory.

[For all other devices]: If executed in a non-secure batch buffer, the address given is in a PPGTT address space. If in a secure ring or batch, the address given is in GGTT space.

13:10 **Reserved**

Format:	MBZ

9 **Flush LLC**

Format:	Enable

If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.

8 **Notify Enable**

Format:	U1

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

7:6 **Reserved**

Format:	MBZ

5:0 **DWord Length**

Format:	=n Total Length - 2

MI_FLUSH_DW

		Value / **Name** 3h — Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]**
1..2	63:48	**Reserved** Format: MBZ
	47:2	**Destination Address** Format: GraphicsAddress[47:2]
	1:0	**Reserved** Format: MBZ
3..4	63:0	**Immediate Data** This field specifies the DWord value to be written to the targeted location. Only valid when 15:14 in header is set to 1h. **Programming Notes** To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FLUSH_DW

MI_FLUSH_DW

Source: VideoCS

Length Bias: 2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:Flush any dirty data to memory. Invalidate the TLB cache inside the hardware Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved**
	21	**Store Data Index** Format: U1 **Description** **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved** Format: MBZ
	18	**TLB Invalidate** Format: U1 If ENABLED, all TLBs belonging to Video Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.
	17	**Reserved** Format: MBZ
	16	**Reserved** Format: MBZ

MI_FLUSH_DW

15:14 Post-Sync Operation

BitFieldDesc

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data	HW implicitly detects the Data size to be Qword or Dword to be written to memory based on the command dword length programmed . When Dword Length indicates Qword, Writes the QWord containing Immediate Data Low, High DWs to the Destination Address . When Dword Length indicates Dword, Writes the DWord containing Immediate Data Low to the Destination Address
2h	Reserved	Reserved
3h		Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.

Programming Notes

If executed in a PPGTT (non-secure) batch buffer, the post-sync op is always inhibited. This command does not write anything to memory.

13:10 Reserved

Format:	MBZ

9 Flush LLC

Format:	Enable

If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.

8 Notify Enable

Format:	U1

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

7 Video Pipeline Cache invalidate

Format:	U1

Enable the invalidation of the video cache at the end of this flush

6 Reserved

Format:	MBZ

5:0 DWord Length

Format:	=n Total Length - 2

MI_FLUSH_DW

		Value / **Name** 3h — Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]**
1..2	63:48	**Reserved** Format: MBZ
	47:3	**Address** Format: GraphicsAddress[47:3]U64 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by the hardware. Upper bits [63:48] are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] is a 4KB page address.
	2	**Destination Address Type** Defines address space of Destination Address **Value** / **Name** / **Description** 0h — PPGTT — Use PPGTT address space for DW write 1h — GGTT — Use GGTT address space for DW write **Programming Notes** Ignored if "No write" is the selected in Operation.
	1:0	**Reserved** Format: MBZ
3..4	31:0	**Immediate Data** This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FORCE_WAKEUP

MI_FORCE_WAKEUP

Source: CommandStreamer

Length Bias: 2

This command is used to communicate Force Wakeup request to PM unit. No functionality is performed by this command when none of the mask bits are set and is equivalent to NOOP. Example for usage model: VCS Ring Buffer: MI_FORCE_WAKEUP (Force Render Awake set to '1') MI_SEMPAHORE_SIGNAL (Signal context id 0xABC to Render Command Streamer) MI_FORCE_WAKEUP (Force Render Awake set to '0') MI_BATCH_BUFFER_START STATE Commands .. .. MI_FORCE_WAKEUP (Force Render Awake set to '1') MI_LOAD_REGISTER_IMMEDIATE (Load register 0x23XX in render command streamer with data 0xFFF) MI_FORCE_WAKEUP (Force Render Awake set to '0') ..... MI_BATCH_BUFFER_END

Programming Notes

This command must not be programmed in the render engine command streamer. Use PIPELINE_SELECT command to wake up media engines.

These commands must be used in strict set then clear order. Do not send 2 wakes in a row or 2 clears in a row.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Dh MI_FORCE_WAKEUP Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:16	**Mask Bits** Format: Mask[15:0] **Programming Notes** Must be set to modify corresponding Bits 15:0. (Mask bits must not be set for reserved bits).
	15:10	**Reserved** Format: MBZ

MI_FORCE_WAKEUP

	9:8	**Reserved** Format: MBZ
	7:5	**Reserved** Format: MBZ
	4:2	**Reserved** Format: MBZ
	1	**Force Render Awake** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: U1 When set, Command Streamer sends message to PM to force awake render engine (next instructions require render engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of render engine (next instructions do not require the render engine to be awake. Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [17] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed.
	1:0	**Reserved** Source: RenderCS Format: MBZ
	0	**Force Media Awake** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: U1 When set, Command Streamer sends message to PM to force awake media engine (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [16] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed.

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Source: CommandStreamer

Length Bias: 2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range).
Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

Programming Notes	Source
Many MMIO bits require the engine to be IDLE prior to updating the value. Command streamer does not implicitly put in a pipeline flush in the cases a MMIO bit requires the engine to be IDLE. In the case there was a 3DPRIMITIVE command or GPGPU_WALKER command without any stalling PIPE_CONTROL, one must be inserted prior to a MI_LOAD_REGISTER_IMMEDIATE that is updating a bit that requires the engine to be IDLE.	RenderCS
When executed from a non-privileged batch buffer, MMIO writes are only allowed to the registers listed in User Mode Non-Privileged Registers for the corresponding engine, any writes targeting the register not listed in the User Mode Non-Privileged Register will convert this command to a NOOP.	
The following addresses should NOT be used for LRIs: 1. 0x8800 - 0x88FF 2. >= 0xC0000 Limited LRI cycles to the Display Engine (0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each LRI.	
Programming an MMIO register is equivalent to programming a non-pipeline state to the hardware and hence an explicit stalling flush needs to be programmed prior to programming this command. However for certain MMIO registers based on their functionality doing an explicit stalling flush is exempted. Listed below are the exempted registers. • 3DPRIM_END_OFFSET - Auto Draw End Offset • 3DPRIM_START_VERTEX - Load Indirect Start Vertex • 3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count • 3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count • 3DPRIM_START_INSTANCE - Load Indirect Start Instance • 3DPRIM_BASE_VERTEX - Load Indirect Base Vertex	RenderCS

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode

MI_LOAD_REGISTER_IMM

	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode
	22:20	**Reserved** Format: MBZ
	19	**Reserved**
	18:13	**Reserved** Format: MBZ
	12	**Reserved**
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range: Must specify a valid register write operation If [11:8] is '1111b', then this command will behave as a NOOP. Otherwise, the value is forwarded to the destination register.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Offset** Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_LOAD_REGISTER_MEM

MI_LOAD_REGISTER_MEM

Source: RenderCS, BlitterCS, VideoCS, VideoEnhancementCS

Length Bias: 2

The MI_LOAD_REGISTER_MEM command requests from a memory location and stores that DWord to a register.

Programming Notes

The command temporarily halts commands that will cause cycles down the 3D pipeline.

The following addresses should NOT be used for MMIO writes:

- 0x8800 - 0x88FF
- >= 0xC0000

Limited MMIO writes cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each MMIO write.

This command should not be used within a non-privilege batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation.

This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 29h MI_LOAD_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean This bit if set when executing from a non-privileged batch buffer will be treated as privilege access violation. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or ring buffer. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Async Mode Enable** Format: Enable If this bit is set then the command stream will not wait for completion of this command before executing the next command. Please refer to the LOAD_INDIRECT and Predicate registers for usage of this bit.

MI_LOAD_REGISTER_MEM

	20	**Reserved**
	19	**Reserved**
	18:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: Excludes DWord (0,1) **[Default]**
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Address** Format: MMIOAddress[22:2] This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.
	1:0	**Reserved** Format: MBZ
2..3	63:2	**Memory Address** Format: GraphicsAddress[63:2] This field specifies the address of the memory location where the register value specified in the DWord above will read from. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Format: MBZ

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_REG

Source: CommandStreamer

Length Bias: 2

The MI_LOAD_REGISTER_REG command reads from a source register location and writes that value to a destination register location.

Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

Programming Notes

The command temporarily halts commands that will cause cycles down the 3D pipeline.

Destination register with mask implemented will not get updated unless the value read from source register has the bits corresponding to the mask bits set. Note that any mask implemented register when read returns "0" for the bits corresponding to mask location. When the source and destination are mask implemented registers, destination register will not get updated with the source register contents.

This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.

Workaround

Workaround: Reads to MMIO registers outside GT (Source Register Address > 0x40000) are not supported.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Ah MI_LOAD_REGISTER_REG Format: OpCode
	22:20	**Reserved** Format: MBZ
	19:18	**Reserved** Format: MBZ
	17:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 1h Format: =n Total Length - 2. Excludes DWord (0,1).

MI_LOAD_REGISTER_REG		
1	31:23	**Reserved** Format: MBZ
	22:2	**Source Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.
	1:0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:2	**Destination Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.
	1:0	**Reserved** Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_EXCL

Source: BlitterCS

Length Bias: 2

The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is *outside* this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe.

Note: The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 13h MI_LOAD_SCAN_LINES_EXCL Format: OpCode
	22	**Reserved** Format: MBZ
	21:19	**Display Pipe Select** Format: U3 This field selects which Display Engine (pipe) this command is targeting. Value / Name: 0h: Display Pipe A 1h: Display Pipe B 2h, 3h: Reserved 4h: Display Pipe C 5h: Reserved 6h, 7h: Reserved
	18:17	**Reserved**
	16:6	**Reserved** Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:16	**Start Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]
	15:0	**End Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_EXCL

Source: RenderCS

Length Bias: 2

The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is **outside** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe.

Note: The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 13h MI_LOAD_SCAN_LINES_EXCL Format: OpCode
	22	**Reserved** Format: MBZ
	21:19	**Display Pipe Select** Format: U3 This field selects which Display Engine (pipe) this command is targeting. Value / Name: 0h Display Pipe A 1h Display Pipe B 2h Reserved 3h Reserved 4h Display Pipe C 5h Reserved
	18:17	**Reserved**
	16:6	**Reserved** Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

	5:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:29	**Reserved** Format: MBZ
	28:16	**Start Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line Window.
	15:13	**Reserved** Format: MBZ
	12:0	**End Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL

Source: BlitterCS

Length Bias: 2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is *within* this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while inside of the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 12h MI_LOAD_SCAN_LINES_INCL Format: OpCode
	22	**Reserved** Format: MBZ
	21:19	**Display Pipe Select** Format: U3 This field selects which Display Engine (pipe) this command is targeting. 0h – Display Pipe A 1h – Display Pipe B 2h, 3h – Reserved 4h – Display Pipe C 5h – Reserved 6h, 7h – Reserved
	18:17	**Reserved**
	16:6	**Reserved** Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_LOAD_SCAN_LINES_INCL		
1	31:16	**Start Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]
	15:0	**End Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL

Source: RenderCS

Length Bias: 2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is **within** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while inside the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 12h MI_LOAD_SCAN_LINES_INCL Format: OpCode
	22	**Reserved** Format: MBZ
	21:19	**Display Pipe Select** Format: U3 This field selects which Display Engine (pipe) this command is targeting. Value / Name: 0h Display Pipe A 1h Display Pipe B 2h Reserved 3h Reserved 4h Display Pipe C 5h Reserved
	18:17	**Reserved**
	16:6	**Reserved** Format: MBZ
	5:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).

MI_LOAD_SCAN_LINES_INCL

1	31	**Reserved** Format: MBZ
	30	**Reserved** Default Value: 1h Format: Must Be One
	29	**Reserved** Format: MBZ
	28:16	**Start Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line window.
	15:13	**Reserved** Format: MBZ
	12:0	**End Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the ending scan line number of the Scan Line Window.

MI_LOAD_URB_MEM

MI_LOAD_URB_MEM

Source: RenderCS

Length Bias: 2

The MI_LOAD_URB_MEM command requests from a memory location and stores that DWord to the URB.

Programming Notes

The command temporarily halts commands that will cause cycles down the 3D pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Ch MI_LOAD_URB_MEM Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: **[Default]**
1	31:15	**Reserved** Format: MBZ
	14:2	**URB Address** This field specifies Bits 14:2 of the URB offset the DWord will be written in the URB. This command only supports writing below 32KB of the URB space.
	1:0	**Reserved** Format: MBZ
2..3	63:6	**Memory Address** Format: GraphicsAddress[63:6] This field specifies the address of the location of where the value will be read from memory. The value must be in the first DW location of the cache line. Range = GraphicsVirtualAddress[47:6] GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	5:0	**Reserved** Format: MBZ

MI_MATH

MI_MATH

Source: BlitterCS

Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer.
This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.
When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Source: VideoCS

Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer.
This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.
When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Source: VideoEnhancementCS

Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer.
This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.
When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Source: RenderCS

Length Bias: 2

The MI_MATH command allows SW to send instruction to ALU in Render Command Streamer.
MI_MATH command is the means by which ALU can be accessed. ALU instructions form the data payload of MI_MATH command, ALU instruction is dword in size. MI_MATH Dword Length should be programmed based on the number of ALU instruction packed, max number is limited by the max Dword Length supported. When MI_MATH command is parsed by command streamer it outputs the payload dwords (ALU instructions) to the ALU. ALU takes single clock to process any given instruction. Refer to B-spec "Command Streamer (CS) ALU Programming" section in Command Streamer Programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_NOOP

MI_NOOP

Source: VideoEnhancementCS

Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value / Name / Description 1 / / Write th NOP_ID Register 0 / / Do not write the NOP_ID register
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP		
Source:	BlitterCS	
Length Bias:	1	
The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).		

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value / Name / Description 0h / Disable / Do not write the NOP_ID register. 1h / Enable / Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP

Source: RenderCS

Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

Performance

The MI_NOOP process time is reduced to 1 clock. An example use of the improved NOOP throughput is for some multi-pass media applications where some unwanted media object commands are replaced by MI_NOOP commands without repacking the commands in a batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified, making this command an effective "no operation" function. Value / Name / Description 0h / Disable / Do not write the NOP_ID register. 1h / Enable / Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP

Source: VideoCS

Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value: 1 — Name: Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_PREDICATE

MI_PREDICATE

Source: RenderCS

Length Bias: 1

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ch MI_PREDICATE Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:6	**Load Operation** This field controls if/how the Predicate state bit is modified. 0h KEEP — The Predicate state bit is unmodified. 1h Reserved 2h LOAD — The Predicate state bit is loaded with the combine operation result. 3h LOADINV — The Predicate state bit is loaded with the inverted combine operation result.
	5	**Reserved** Format: MBZ
	4:3	**Combine Operation** This field controls if/how the result of the compare operation is combined with the current Predicate state bit. 0h SET — The combine operation output the compare result unmodified. 1h AND — The combine operation outputs the AND of the compare result and the current Predicate state bit. 2h OR — The combine operation outputs the OR of the compare result and the current Predicate state bit. 3h XOR — The combine operation outputs the XOR of the compare result and the current Predicate state bit.
	2	**Reserved** Format: MBZ

MI_PREDICATE

	1:0	**Compare Operation** This field controls how Data DWord 0 and Data DWord 1 fields are used to generate a compare operation result and possibly modify the PredicateData register.

Value	Name	Description
0h	TRUE	The compare operation outputs TRUE. The PredicateData register is unmodified.
1h	FALSE	The compare operation outputs FALSE. The PredicateData register is unmodified.
2h	SRCS_EQUAL	(MItemp0 - MItemp1) is computed and loaded into the PredicateData register. The compare operation outputs (MItemp0 == MItemp1).
3h	DELTAS_EQUAL	(MItemp0 - MItemp1) is computed and compared to the PredicateData register. If the values are equal, the compare operation outputs TRUE, otherwise it outputs FALSE. The PredicateData register is unmodified.

MI_REPORT_HEAD

MI_REPORT_HEAD

Source: VideoEnhancementCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.

When the **Per-Process Virtual Address Space and Execlist Enable bit** is reset: The location written is relative to the address programmed in the Hardware Status Page Address Register. When the **Execlist Enable** is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source: BlitterCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.
When the **Execlist Enable** bit is reset:
The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).
When the **Execlist Disable** is clear, the head pointer will be reported to the PP HW Status Page.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD
	22:0	**Reserved** Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source: RenderCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location. When Execlist Enable is set, the head pointer will be reported to the PP HW Status Page. The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes

This command must not be executed from a Batch Buffer. (Refer to the description of the HWS_PGA register.)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source: VideoCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.When the Per-Process Virtual Address Space and Execlist Enable bitis reset:The location written is relative to the address programmed in the Hardware Status Page Address Register. When the Execlist Enable is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_REPORT_PERF_COUNT

MI_REPORT_PERF_COUNT

Source: RenderCS

Length Bias: 2

The MI_REPORT_PERF_COUNT command causes the GFX hardware to write out a snap-shot of performance counters to the address specified in this command along with constant ID field supplied and the time-stamp counter. This write is required to be treated as a cacheable write irrespective of GTT entry memory type. This command is specific to the render engine.

Programming Notes

GTT_SELECT must not be set to 1 (i.e. GGTT) when MI_REPORT_PERF_COUNT command is programmed in a non-privileged batch buffer. Refer to the "User Mode Privileged commands" Table in MI_BATCH_BUFFER_START command section for more details.
All batch buffers in PPGTT are considered as Non-privileged.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 28h MI_REPORT_PERF_COUNT Format: OpCode
	22:6	**Reserved** Format: MBZ
	5:0	**DWord Length** Format: =n Total Length - 2 Value: 2h — Name: Excludes DWord (0,1) **[Default]**
1..2	63:6	**Memory Address** Format: GraphicsAddress[63:6] This field specifies 64B aligned GFX MEM address where the chap counter values are reported. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47] **Programming Notes** This field is ignored if "Report to OABUFFER" bit is set.
	5	**Reserved** Format: MBZ

MI_REPORT_PERF_COUNT

	4	**Core Mode Enable** Format: U1 This bit is set then the address will be offset by the Core ID:If Core ID 0, then there is no offsetIf Core ID 1, then the Memory is offset by the size of the data(64b).
	3:1	**Reserved** Format: MBZ
	0	**Use Global GTT** Format: Boolean This field when set (i.e. bit = 1) selects the GGTT for address translation. When this bit is 0 (default value), HW should use PGTT for address translation.
3	31:0	**Report ID** Format: U32 This field specifies the ID provided by SW for a given report command. It can be tracked to use different flavors of these reports based on where in command-stream they are inserted.This field is reported only when Counter Select Field is 0. **Programming Notes** If a privilege access violation occurs, the REPORT ID field in the report generated by the next legitimate MI_REPORT_PERF_COUNT will be corrupted.

MI_RS_CONTEXT

MI_RS_CONTEXT

Source: RenderCS

Length Bias: 1

The MI_RS_CONTEXT command is used to force a resource streamer context save or restore.

Programming Notes

This command must not be used/programmed in Execution List mode of scheduling.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Fh MI_RS_CONTEXT Format: OpCode
	22:1	**Reserved** Format: MBZ
	0	**Resource Streamer Save** Format: U1 This bit specifies whether the MI_RS_CONTEXT command will cause the resource streamer context to be saved or restored. Value / Name / Description 0h / Restore / Resource Streamer context is restored 1h / Save / Resource Streamer context is saved

MI_RS_CONTROL

MI_RS_CONTROL

Source: RenderCS

Length Bias: 1

The MI_RS_CONTROL command is used to start or stop the Resource Streamer.

Programming Notes

- This command must be programmed only inside a Resource Streamer enabled batch buffer.
- This command provides means to selectively disable or enable Resource Streamer for set of commands in a Resource Streamer enabled batch buffer
- On re-enabling the Resource Streamer through this command, command streamer will start Resource Streamer on the next non-sync command of the batch buffer.
- This command status is render context save/restored during context switching.
- The scope of MI_RS_CONTROL is within the batch buffer it is programmed, it doesn't get carried to the following chained batch buffer or second level batch buffer. RS control status goes back to default mode of Resource Streamer Enabled on all batch buffer arbitration boundaries. Batch buffer arbitration boundaries includes calling a chained or a second level batch buffer through MI_BATCH_BUFFER_START command or terminating a batch buffer through MI_BATCH_BUFFER_END command.
- Example:

1. MI_BATCH_START (Primary batch buffer with RS enable)
2. Command 1 --> CS starts RS
3. Command 2
4. :
5. MI_RS_CONTROL (stop option) -> RS will stop on this command, CS sets RS control status to STOP.
6. Command 3
7. MI_BATCH_START (2nd level batch with RS enable not set, RS control status gets reset to default status of START)
8. :
9. MI_BATCH_END (Second Level Batch End)
10. Command 4 --> CS starts RS here as RS control flag gets reset to START at step-7
11. MI_BATCH_BUFFER_END

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode

MI_RS_CONTROL

	28:23	**MI Command Opcode** Default Value: 06h MI_RS_CONTROL Format: OpCode
	22:1	**Reserved** Format: MBZ
	0	**Resource Streamer Control** Format: U1 This bit specifies whether the command is starting or stopping the Resource Streamer.

Value	Name	Description
0h	Stop	Stop and disable the Resource Streamer
1h	Start	Start and enable the Resource Streamer

MI_RS_STORE_DATA_IMM

MI_RS_STORE_DATA_IMM

Source: RenderCS

Length Bias: 2

The MI_RS_STORE_DATA_IMM command requests a write of the DWord constant supplied in the packet to the specified Memory Address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Bh Format: OpCode MI_RS_STORE_DATA_IMM
	22	**Reserved** Format: MBZ
	21	**Reserved**
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Format: =n Total Length - 2. Excludes DWord (0,1).
1..2	63:2	**Destination Address** Format: GraphicsAddress[63:2] This field specifies Bits 47:2 of the Address where the DWord will be stored. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. When render engine is PPGTT enabled this Address is translated using PPGTT, else GGTT is used for translation.
	1	**Reserved** Format: MBZ
	0	**Core Mode Enable** If this bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data.

MI_RS_STORE_DATA_IMM		
3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_SIGNAL

Source: CommandStreamer

Length Bias: 2

Description

This command is used to signal the target engine stating the memory semaphore update occurrence to one of its contexts with **Target Context ID.** MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command. MI_ATOMIC (non-posted) command will be programmed prior to this command to update the semaphore data in memory.

Workaround

Workaround: Post-Sync operation bit must not be set when Target Engine Select is set to RCS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Bh MI_SEMAPHORE_SIGNAL Format: OpCode
	22	**Reserved** Format: MBZ
	21	**Post-Sync Operation** Source: RenderCS

Value	Name	Description
0h	No Post Sync Operation	Command is executed as usual.
1h	Post Sync Operation	MI_SEMAPHORE_SIGNAL command is executed as a pipelined PIPE_CONTROL flush command with Semaphore Signal as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding flushes issued prior to this command.

Programming Notes

Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command.

When this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following

MI_SEMAPHORE_SIGNAL

		commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued. When this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.
	21	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	20:19	**Reserved** Format: MBZ
	18	**Reserved** Format: MBZ
	17:15	**Target Engine Select** This field selects the target engine to which SIGNAL will be send to. Value / Name: 0h RCS 1h VCS0 2h BCS 3h VECS 4h VCS1 5h,6h,7h Reserved
	14:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_SEMAPHORE_SIGNAL		
1	31:0	**Target Context ID** **Description** In execlist based scheduling this field contains the Context ID corresponding to the context of the target engine that this command is signaling. Target engine waiting on MI_SEMPHORE_WAIT in signal mode will re-fetch the data from memory or comparison if its context ID is same as this signaled Context ID. When execlists are enabled, Target engine on receiving this Context ID sends message to the SHIM if it doesn't have the context with the same Context ID running. Message send to SHIM carries the Context ID which will be looked at by UC for rescheduling the signaled Context ID. Target engine waiting on MI_SEMAPHORE_WAIT in signal mode will fetch data from memory for comparison on receiving signal irrespective of the context id received. In ring buffer mode of scheduling this field doesn't have any relevance.

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Source: CommandStreamer

Length Bias: 2

Description

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT. Running on the same engine is only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait. Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in CTXT_SR_CTL register.
- In ring buffer mode of scheduling or Execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- Exec-List Scheduling: CS generates semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Workaround

Workaround: [All Command Streamers][Ring Buffer Mode of Scheduling]: MI_SEMAPHORE_WAIT command must be always programmed with “Wait Mode” set to “Polling Mode” Or MI_SEMAPHORE_WAIT command with “Wait Mode” set to “Polling Mode” can be programmed when “Semaphore Wait Event IDLE message Disable” bit in “RC_PSMI_CTRL” register is set to disable Idle messaging on unsuccessful MI_SEMPAHORE_WAIT.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be 1 if the **Per Process GTT Enable** bit is clear.

MI_SEMAPHORE_WAIT

Value	Name	Description
0h	Per Process Graphics Address	
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

21:18 Reserved

Format:	MBZ

17 Reserved

Format:	MBZ

16 Reserved

15 Wait Mode

This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out.

Value	Name	Description
1h	Polling Mode	In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register.
0h	Signal Mode	In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.

14:12 Compare Operation

This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait.

SAD = Semaphore Address Data
SDD = Semaphore Data Dword

Value	Name	Description
0h	SAD_GREATER_THAN_SDD	If Indirect fetched data is greater than inline data then continue.
1h	SAD_GREATER_THAN_OR_EQUAL_SDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	SAD_LESS_THAN_SDD	If Indirect fetched data is less than inline data then continue.
3h	SAD_LESS_THAN_OR_EQUAL_SDD	If Indirect fetched data is less than or equal to inline data then continue.
4h	SAD_EQUAL_SDD	If Indirect fetched data is equalto inline data then continue.

MI_SEMAPHORE_WAIT

		5h	SAD_NOT_EQUAL_SDD	If Indirect fetched data is not equal to inline data then continue.
		6h	Reserved	
		7h	Reserved	
	11:8	**Reserved**		
		Format:	MBZ	
	7:0	**DWord Length**		
		Format:	=n Total Length - 2	
		Value	**Name**	
		2h	Excludes DWord (0,1) **[Default]**	
1	31:0	**Semaphore Data Dword**		
		Format:	U32	
		This Data dword is supplied by software to control execution of the command buffer. This value is used as part of the comparison to result in waiting or continuing in the command parser if enabled.		
2..3	63:2	**Semaphore Address**		
		Format:	GraphicsAddress[63:2]	
		This field is the Graphics Memory Address of the 32-bit value for the semaphore. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form.		
	1:0	**Reserved**		
		Format:	MBZ	

MI_SET_CONTEXT

MI_SET_CONTEXT

Source: RenderCS

Length Bias: 2

The MI_SET_CONTEXT command is used to specify the logical context associated with the hardware context. A logical context is an area in memory used to store hardware context information, and the context is referenced via a 2KB-aligned pointer. If the (new) logical context is different (i.e., at a different memory address), the device saves the current HW context values to the current logical context address, and then restores (loads) the new logical context by reading the context from the new address and loading it into the hardware context state. If the logical context address specified in this command matches the current logical context address, this command is effectively treated as a NOOP. Specific to the Render command stream only. This command also includes some controls over the context save/restore process. The Force Restore bit can be used to refresh the on-chip device state from the same memory address if the indirect state buffers have been modified. The Restore Inhibit bit can be used to prevent the new context from being loaded at all. This must be used to prevent an uninitialized context from being loaded. Once software has initialized a context (by setting all state variables to initial values via commands), the context can then be stored and restored normally. When switching from a generic media context to a 3D context, the generic media state must be cleared via the Generic Media State Clear bit 16 in PIPE_CONTROL (or bit 4 in MI_FLUSH) before saving 3D context. MI_SET_CONTEXT commands are permitted only within a ring buffer (not within a batch buffer).
All context is saved and restored from a GGTT space.
This command does not initiate any interrupt due to context switch of any kind and does not support any workaround batch buffer or indirect context offset feature.

Programming Notes

This command is legal only if Execlist Enable in the GFX_MODE register is reset. Otherwise, execlists must be used to switch context in lieu of MI_SET_CONTEXT.

For ring buffer mode, the first 128B(2 cache lines) of the context image are saved as zeros.

This command needs to be always followed by a single MI_NOOP instruction to workaround a silicon issue.

MI_ARB_ON_OFF with 'Arbitration Enable Reset' set should be programmed before an MI_SET_CONTEXT command. MI_ARB_ON_OFF with 'Arbitration Enable' set should be programmed after an MI_SET_CONTEXT command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 18h MI_SET_CONTEXT Format: OpCode
	22:8	**Reserved** Format: MBZ

MI_SET_CONTEXT

	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:12	**Logical Context Address** Format: GraphicsAddress[31:12]LogicalContext **Description** This field contains the 4KB-aligned graphics memory address of the Logical Context that is to be loaded into the hardware context. If this address is equal to the CCID register associated with the current ring, no load will occur. Prior to loading this new context, the device will save the existing context as required. After the context switch operation completes, this address will be loaded into the associated CCID register. This field needs to be 4KB aligned virtual address.
	11:10	**Reserved** Format: MBZ
	9	**Reserved** Format: MBZ
	8	**Reserved, Must be 1** Format: Must Be One
	7:5	**Reserved** Format: MBZ
	4	**Core Mode Enable** Format: Enable If set the Context Image will be offset based off the Core ID: If Core ID 0, no offset If Core ID 1, 36KB Offset
	3	**Resource Streamer State Save Enable** Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is saved as part of switching away from this logical context. This bit will be stored in the associated CCID register to control the context save operation when switching away from this context (as part of a subsequent MI_SET_CONTEXT command).

MI_SET_CONTEXT		
	2	**Resource Streamer State Restore Enable** Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is loaded (or restored) as part of switching to this logical context. This bit affects the switch (if required) to the context specified in Logical Context Address. This bit will also be stored in the associated CCID register to control a subsequent context save operation when switching to this context (as part of a subsequent ring buffer switch).
	1	**Force Restore** When switching to this logical context a comparison between Logical Context Address and the contests of the CCID register is performed. Normally, matching addresses prevent a context restore from occurring; however, when this bit is set a context restore is forced to occur. This bit cannot be set with Restore Inhibit. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.
	0	**Restore Inhibit** If set, the restore of the HW context from the logical context specified by Logical Context Address is inhibited (i.e., the existing HW context values are maintained). This bit must be used to prevent the loading of an uninitialized logical context. If clear, the context switch proceeds normally. This bit cannot be set with Force Restore. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.

MI_SET_PREDICATE

MI_SET_PREDICATE

Source: RenderCS

Length Bias: 1

Description

This command sets the Predication Check for the subsequent commands in the command buffer except for MI_SET_PREDICATE itself. Render Command Streamer NOOPs the following commands based on the PREDICATE_ENABLE from MI_SET_PREDICATE, MI_SET_PREDICATE_RESULT and MI_SET_PREDICATE_RESULT_2 status. Resource Streamer doesn't take any action of parsing MI_SET_PREDICATE, this command is similar to any other command which is not meant for resource streamer.

Executing MI_SET_PREDICATE command sets PREDICATE_ENABLE bits in MI_MODE register, MI_MODE register gets render context save restored.

Programming Notes

- MI_SET_PREDICATE predication scope must be confined within a Batch Buffer to set of commands.
- MI_SET_PREDICATE with Predicate Enable Must always have a corresponding MI_SET_PREDICATE with Predicate Disable within the same Batch Buffer.
- MI_ARB_CHK command must be programmed outside the Predication Scope of MI_SET_PREDICATE.
- MI_SET_PREDICATE Predication Scope must not involve any RC6 triggering events.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 01h MI_SET_PREDICATE Format: OpCode
	22:4	**Reserved** Format: MBZ
	3:0	**PREDICATE ENABLE** This field sets the predication logic in render command streamer when parsed. Predicate Disable is the default mode of operation. Value / Name / Description 0h / NOOP Never / Predication is Disabled and RCS will process commands as usual. 1h / NOOP on Result2 clear / Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is clear.

MI_SET_PREDICATE

		2h	NOOP on Result2 set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is set.
		3h	NOOP on Result clear	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is clear.
		4h	NOOP on Result set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is set.
		5h	Execute when one slice enabled.	Following Commands will be Executed by RCS only when one slice is enabled.
		6h	Execute when two slices are enabled.	Following Commands will be Executed by RCS only when two slices are enabled.
		7h	Execute when three slices are enabled.	Following Commands will be Executed by RCS only when all the three slices are enabled.
		8h-Ah	Reserved	
		Bh, Ch	Reserved	
		Dh, Eh	Reserved	
		Fh	NOOP Always	Following Commands will be NOOPED by RCS unconditionally.

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Source: CommandStreamer

Length Bias: 2

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

This command supports writing to multiple consecutive dwords or qwords memory locations from the starting address.

Programming Notes

- This command should not be used within a "non-privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Format: Boolean If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.

MI_STORE_DATA_IMM

	21	**Store Qword** Format: Boolean If set, this command generates Qword writes to memory, two "Data Dword" are paired to form a Qword. Number of qwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x/2" qword writes to memory. If reset this command generates Dwords writes to memory. Number of dwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x" dword writes to memory.
	20:10	**Reserved** Format: MBZ
	9:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1) **Value** / **Name** 2h — Store Dword **[Default]** 3h — Store Qword **Programming Notes** / **Source** DWord Length programmed must not exceed 0x3FE. — If RS is enabled in the batch buffer, then the value of this field must not exceed 0x3F. — RenderCS
1..2	63:48	**Reserved** Format: MBZ
	47:2	**Address** Format: GraphicsAddress[47:2] GraphicsAddress is 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost bits are ignored and MBZ. This field specifies Bits 47:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.
	1	**Reserved** Format: MBZ
	0	**Core Mode Enable** Format: U1 This bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data(32b or 64b based off number of DW length).

MI_STORE_DATA_IMM		
3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Source: VideoEnhancementCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Format: MBZ
	21	**Use Per-Process Hardware Status Page** If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Format: MBZ

MI_STORE_DATA_INDEX

	11:2	**Offset** Format: U10 Zero-based DWord offset into the HW status page Format: GraphicsAddress[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED. For a QWord write, the offset is valid down to bit 3 only. **Value**: [16, 1023] **Name**:
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).
3	31:0	**Data Word 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Source: BlitterCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED. This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers). This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX
	22	**Reserved** Format: MBZ
	21	**Use Per-Process Hardware Status Page** If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Format: MBZ
	11:2	**Offset** Format: U10 zero-based DWord offset into the HW status page. Format: HardwareStatusPageOffset[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written.

MI_STORE_DATA_INDEX

		Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED.This address must be 8B aligned for a store "QW" command. **Value** / **Name**: [16, 1023] /
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Source: RenderCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved**
	21	**Use Per-Process Hardware Status Page** If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be 0 if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 1h Format: =n Total Length - 2. Excludes DWord (0,1) = 1 for DWord, 2 for QWord.
1	31:12	**Reserved** Format: MBZ

MI_STORE_DATA_INDEX

	11:2	**Offset** Format: U10 zero-based DWord offset into the HW status page. Format: HardwareStatusPageOffset[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED. This address must be 8B aligned for a store QW command. **Value**: [16, 1023] **Name**:
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Source: VideoCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Format: MBZ
	21	**Use Per-Process Hardware Status Page** If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Format: MBZ

MI_STORE_DATA_INDEX

	11:2	**Offset** Format: U10 zero-based DWord offset into the HW status page Format: GraphicsAddress[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. For a QWord write, the offset is valid down to bit 3 only. **Value** / **Name** [16, 1023] **Programming Notes** The first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED.
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).
3	31:0	**Data Word 1** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM

Source: CommandStreamer

Length Bias: 2

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes

- The command temporarily halts command execution.
- The memory address for the write is snooped on the host bus.
- This command should not be used from within a "non-privilege" batch buffer to access global virtual space. doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or "privilege" batch buffers to access global virtual space.
- This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers.

Workaround

Workaround: Reads to MMIO registers outside GT (Register Address > 0x40000) are not supported.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean It is allowed for this bit to be set when executing this command from a privileged (secure) batch or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_STORE_REGISTER_MEM

	21	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	21	**Predicate Enable** Source: RenderCS Format: U1 If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.
	20	**Reserved** Format: MBZ
	19	**Reserved**
	18:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** 2h: Excludes DWord (0,1) **[Default]**
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. **Programming Notes** • Storing a VGA register is not permitted and will store an UNDEFINED value. • The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified.
	1:0	**Reserved** Format: MBZ

MI_STORE_REGISTER_MEM		
2..3	63:2	**Memory Address** Format: GraphicsAddress[63:2]MMIO This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data.Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Format: MBZ

MI_STORE_URB_MEM

MI_STORE_URB_MEM

Source: RenderCS

Length Bias: 2

The MI_STORE_URB_MEM command requests a URB read from a specified memory mapped URB location in the device and store of that DWord to memory. The URB address is specified along with the command to perform the read.

Programming Notes

- The command temporarily halts command execution

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Dh MI_STORE_URB_MEM Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: **[Default]**
1	31:15	**Reserved** Format: MBZ
	14:2	**URB Address** This field specifies Bits 14:2 of the URB offset the DWord will be read in the URB. This command only supports reading from the lower 32KB of the URB space.
	1:0	**Reserved** Format: MBZ

MI_STORE_URB_MEM

2..3	63:6	**Memory Address** Format: GraphicsAddress[63:6] This field specifies the address of the location of where the value will be written to memory. The value must be in the first DW location of the cache line. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	5:0	**Reserved** Format: MBZ

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source:	VideoEnhancementCS	
Length Bias:	1	
Description		
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable **Description** This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source: BlitterCS Length Bias: 1		
Description		
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable **Description** This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Source: RenderCS

Length Bias: 1

Description
Blocks PM Flush Requests.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable **Description** This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source:	VideoCS	
Length Bias:	1	
Description		
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable **Description** This field suspends flush due to a PM flush request.

MI_TOPOLOGY_FILTER

MI_TOPOLOGY_FILTER

Source: RenderCS

Length Bias: 1

This command is used to specify a specific 3DPrimType value, where the CS will ignore all 3DPRIMITIVE commands that do no have a matching 3DPrimType. This primitive culling is optional (turned off by using this command with a Topology Filter Value of 0). **This command is specific to the Render command stream only.**

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Dh MI_TOPOLOGY_FILTER Format: OpCode
	22:6	**Reserved** Format: MBZ
	5:0	**Topology Filter Value** Format: 3D_Prim_Topo_Type When non-zero, the CS will discard all 3DPRIMITIVE commands which do not match the specified 3DPrimTopologyType. When zero, no filtering is performed (normal operation).

MI_UPDATE_GTT

MI_UPDATE_GTT		
Source:	BSpec	
Length Bias:	2	
Description		**Source**
The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow. A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries. PIPE_CONTROL flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering). MI_UPDTE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer. PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.		RenderCS
The MI_UPDATE_GTT command is used to update GGTT page table entries in a coherent manner and at a predictable place in the command flow. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries. MI_FLUSH_DWORD flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering). MI_UPDATE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer. PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.		BlitterCS, VideoCS, VideoEnhancementCS
DWord	**Bit**	**Description**

MI_UPDATE_GTT

0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved** Format: MBZ
	9:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:12	**Entry Address** Format: GraphicsAddress[31:12] This field holds the QW offset of the first table entry to be modified in GGTT.
	11:0	**Reserved** Format: MBZ
2..n	63:0	**Entry Data** Format: PageTableEntry This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

MI_URB_ATOMIC_ALLOC

MI_URB_ATOMIC_ALLOC

Source: RenderCS

Length Bias: 1

This command is used to specify the region in URB allocated for URB atomic value storage.
This command is specific to the Render command stream only.

Programming Notes

This command can only be sent after a flush has occurred.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 09h MI_URB_ALLOC Format: OpCode
	22:20	**Reserved** Format: MBZ
	19:12	**URB Atomic Storage Offset** Format: U8 Number of 128B Entries This field specifies the offset of a 128B granular starting address in the URB. The value of **URB Atomic Storage Offset** plus the value of the **URB Atomic Storage Size** must not exceed 256. Value: [0,255] — Name: — Description: 0-(32KB-128B)
	11:9	**Reserved** Format: MBZ
	8:0	**URB Atomic Storage Size** Format: U9 Number of 128B Entries This field specifies the size of the buffer in the URB in number of 128B entries. If this field has a value of zero then the URB Atomic allocation is disabled and will not be context save/restored. Value: [0,256] — Name: — Description: 0-32KB

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source: VideoEnhancementCS

Length Bias: 1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT		
Source: BlitterCS Length Bias: 1		
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT
	22:0	**Reserved** Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source: RenderCS

Length Bias: 1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT		
Source: VideoCS Length Bias: 1		
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT

Source: RenderCS, BlitterCS

Length Bias: 1

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will effectively give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

Execution List Mode of Scheduling: CS on evaluating MI_WAIT_FOR_EVENT to be unsuccessful (has to wait for event to happen) triggers synchronous context switch stating the switch reason in Context Status Buffer. Note that synchronous context switch can be inhibited through programming “Inhibit Synchronous Context Switch” bit in CTXT_SR_CTL register or by disabling arbitration through MI_ARB_ON_OFF command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22	**Reserved** Format: MBZ

MI_WAIT_FOR_EVENT

	21	**Display Plane 1 C Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 C "Vertical Blank" event occurs. This event is described as the start of the next Display C vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event in the Device Programming Interface chapter of MI Functions.
	20	**Display Plane 6 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	19	**Display Plane 12 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	18	**Display Plane 11 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	17	**Display Plane 10 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

	16	**Display Plane 9 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	15	**Display Plane 3 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 1 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	14	**Display Plane 1 C Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 C "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe C Display Scan Line Count Range Compare Register.
	13:12	**Reserved** Format: MBZ
	11	**Display Plane 1 B Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 B "Vertical Blank" event occurs. This event is described as the start of the next Display B vertical blank period. Note that this can cause a wait for up to an entire refresh period.
	10	**Display Plane 5 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

	9	**Display Plane 2 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	8	**Display Plane 1 B Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 B "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register.
	7	**Display Plane 8 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	6	**Display Plane 7 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	5:4	**Reserved** Format: MBZ
	3	**Display Plane 1 A Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 A "Vertical Blank" event occurs. This event is described as the start of the next Display A vertical blank period. Note that this can cause a wait for up to an entire refresh period.

MI_WAIT_FOR_EVENT

	2	**Display Plane 4 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	1	**Display Plane 1 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 1 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	0	**Display Plnae 1 A Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 A "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register.

Move

mov - Move

Source: Eulsa

Length Bias: 4

The mov instruction moves the components in src0 into the channels of dst. If src0 and dst are of different types, format conversion is performed. If src0 is a scalar immediate, the immediate value is loaded into enabled channels of dst.

A mov with the same source and destination type, no source modifier, and no saturation is a raw move. A packed byte destination region (B or UB type with HorzStride == 1 and ExecSize > 1) can only be written using raw move.

When denorm mode is flush to zero, a raw mov instruction with saturation modifier will not flush the denorm input or output to zero (Denorm is preserved).

Format: [(pred)] mov[.cmod] (exec_size) dst src0

Programming Notes

A *mov* instruction with a source modifier always copies a denorm source value to a denorm destination value (in the manner of a raw move).

There is no direct conversion from B/UB to DF or DF to B/UB. Use two instructions and a word or DWord intermediate type.

There is no direct conversion from B/UB to Q/UQ or Q/UQ to B/UB. Use two instructions and a word or DWord intermediate integer type.

There is no direct conversion from HF to DF or DF to HF. Use two instructions and F (Float) as an intermediate type.

There is no direct conversion from HF to Q/UQ or Q/UQ to HF. Use two instructions and F (Float) or a word integer type or a DWord integer type as an intermediate type.

Restriction

Restriction : Raw move is not supported for Float values in ALT mode if any values are infinities or NaNs.

Restriction : An accumulator can be a source or destination operand but not both.

Syntax

```
[(pred)] mov[.cmod] (exec_size) reg reg [(pred)] mov[.cmod] (exec_size) reg imm32
[(pred)] mov[.cmod] (exec_size) reg imm64
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) {
dst.chan[n] = src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

mov - Move

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*B,*W,*D	F
F	*B,*W,*D
F	F
*W,*D	DF
F	DF
DF	*W,*D
DF	F
DF	DF
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q
F	*Q
DF	*Q
*Q	F
*Q	DF
*B,*W,*D	HF
F	HF
HF	*B,*W,*D
HF	F
HF	HF

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Move Indexed

movi - Move Indexed

Source: Eulsa

Length Bias: 4

The movi instruction performs a fast component-wise indexed move for subfields from src0 to dst. The source operand must be an indirectly-addressed register. All channels of the source operand share the same register number, which is provided by the register field of the first address subregister, with a possible immediate register offset. The register fields of the subsequent address subregisters are ignored by hardware. The subregister number of a source channel is provided by the subregister field of the corresponding address subregister, with a possible immediate subregister offset.
The destination register may be either a directly-addressed or an indirectly-addressed register.
This instruction effectively performs a subfield shuffling from one register to another. Up to eight subfields can be selected by an instruction.

Format: [(pred)] movi (exec_size) dst src0

Programming Notes

HW Implementation Details:
The source register is calculated by adding the register portion of the first index register with the register portion of the address immediate, a0.0[11:5] + addr_imm[9:5]
For byte movi, byte0 of the destination is selected by (a0.0[4:0]), byte1 is selected by (a0.1[4:0]), ..., and byte7 is selected by (a0.7[4:0]). The rest of the bytes are undefined.
For word movi, byte0 of the destination is selected by (a0.0[4:1] & 0), byte1 is selected by (a0.0[4:1] & 1), byte2 is selected by (a0.1[4:1] & 0), byte3 is selected by (a0.1[4:1] & 1), ..., and byte15 is selected by (a0.7[4:1] & 1). The rest of the bytes are undefined.
For DWord or float movi, byte0 of the destination is selected by (a0.0[4:2] & 00b), byte1 is selected by (a0.0[4:2] & 01b), byte2 is selected by (a0.0[4:2] & 10b), byte3 is selected by (a0.0[4:2] & 11b), byte4 is selected by (a0.1[4:2] & 00b), byte5 is selected by (a0.1[4:2] & 01b), ..., byte31 is selected by (a0.7[4:2] & 11b).
For all 3 conditions above, a0.n[4:0] = a0.n[4:0] + addr_imm[4:0].

Restriction

Restriction : Source operand cannot be accumulators. The source operand must be a general register.

Restriction : The source and destination must have the same type.

Restriction :
The address register for the source must be aligned to the base (a0.0).

Restriction : The destination register (directly or indirectly addressed) must be 16-byte aligned.

Restriction : The destination region (directly or indirectly addressed) must point to the same GRF register.

Restriction : The destination stride in bytes must equal the source element size in bytes.

Restriction : The Align16 access mode is not allowed.

Restriction : All the index registers (address subregisters) used must point to the same GRF register.

Restriction : The instruction must use 1x1 indirect regioning.

movi - Move Indexed

Restriction : The destination offset is only used to create channel enables. Each element of the destination is directly mapped to the index registers for the movi instruction. i.e. a0.0 -> dst.0, a0.1 -> dst.1, a0.2 -> dst.2, etc.

Restriction :
Only 8 address subregisters are used (a0.0-a0.7). Destination element 8 will be sourced from address register zero (a0.0), dst.9 <-a0.1, etc. This is an exception to the above restriction, for example:
movi (8) r31.8:uw r[a0.0,0]<8;8,1>:uw // r31.8:uw<-a0.0:uw, r31.9:uw<-a0.1:uw, etc.

Restriction : Conditional Modifier is not allowed for this instruction.

Syntax

```
[(pred)] movi (exec_size) reg reg imm
```

Pseudocode

```
Evaluate(WrEn); srcregfile = regfile(src0); srcregbase = reg(address[0]) +
reg(addr_imm); for ( n = 0; n < RegWidth; n++ ) { if ( WrEn.chan[n] ) { srcsubreg
= subreg(address[n] + addr_imm); dst.chan[n] = srcregfile.srcreg.srcsubreg; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
B	B
UB	UB
W	W
UW	UW
D	D
UD	UD
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Multiply

mul - Multiply

Source: Eulsa

Length Bias: 4

The mul instruction performs component-wise multiplication of src0 and src1 and stores the results in dst. When multiplying integer datatypes, if src0 is DW and src1 is W, irrespective of the destination datatype, the accumulator maintains full 48-bit precision. This is required to handle the macro for 32x32 multiplication. The macro described in the mach instruction should be used to obtain the full precision 64-bit multiplication results.

Note: A 32x32 multiply operation is handled natively, without a macro. When operating in this mode, the resulting 64-bit data is packed, unlike the macro, where the lower and upper 32 bits of the result are written to different general registers by two separate instructions. Refer to the macro description for details.

When multiplying integer data types, if one of the sources is a DW, the resulting full precision data is stored in the accumulator. However, if the destination data type is either W or DW, the low bits of the result are written to the destination register and the remaining high bits are discarded. This results in undefined Overflow and Sign flags. Therefore, conditional modifiers and saturation (.sat) cannot be used in this case.

Format:

[(pred)] mul[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Integer source operands cannot be accumulators.

Restriction : When multiplying a DW and any lower precision integer, the DW operand must on src0.

Syntax

```
[(pred)] mul[.cmod] (exec_size) reg reg reg [(pred)] mul[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] * src1.chan[n];
     }
 }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B	*B
*B	*W
*B	*D

mul - Multiply

*W	*W
*W	*D
*D	*D
*D	*Q
F	F
DF	DF
HF	HF
HF, F	HF, F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Multiply Accumulate

mac - Multiply Accumulate

Source: Eulsa

Length Bias: 4

The mac instruction takes component-wise multiplication of src0 and src1, adds the results with the corresponding accumulator values, and then stores the final results in dst.

Format:
[(pred)] mac[.cmod] (exec_size) dst src0 src1

Programming Notes

When source and destination datatypes are different, the implied datatype for the accumulator operand is always the destination datatype.

Restriction

Restriction : Accumulator is an implicit source and thus cannot be an explicit source operand.

Syntax

```
[(pred)] mac[.cmod] (exec_size) reg reg reg [(pred)] mac[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
   for ( n = 0; n < exec_size; n++ ) {
       if ( WrEn.chan[n] ) {
           dst.chan[n] = src0.chan[n] * src1.chan[n] + acc0.chan[n];
       }
   }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W	*B,*W,*D
F	F
DF	DF
HF	HF
HF, F	HF, F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

mac - Multiply Accumulate

	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Multiply Accumulate High

mach - Multiply Accumulate High

Source: Eulsa

Length Bias: 4

The mach instruction performs DWord integer multiply-accumulate operation and outputs the high DWord (bits 63:32). For each enabled channel, this instruction multiplies the DWord in src0 with the high word of the DWord in src1, left shifts the result by 16 bits, adds it with the corresponding accumulator values, and keeps the whole 64-bit result in the accumulator. It then stores the high DWord (bits 63:32) of the results in dst. This instruction is intended to be used to emulate 32-bit DWord integer multiplication by using the large number of bits available in the accumulator. For example, the following instructions perform vector multiplication of two 32-bit signed integer sources from r2 and r3 and store the resulting vectors with the high 32 bits in r5 and the low 32 bits in r6.

```
mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw
mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d
mov (8) r6.0<1>:d acc0:d // Low 32 bits.
```

Here is a different example including negation. An added preliminary mov is required for source modification on src1.

```
mov (8) r3.0<1>:d -r3<8;8,1>:d
mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw
mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d // High 32 bits
mov (8) r6.0<1>:d acc0:d // Low 32 bits.
```

The mach should have channel enable from the destHI of IMUL, the mov should have the channel enable from the destLO of IMUL. As mach is used to generate part of the 64-bit DWord integer results, saturation modifier should not be used. In fact, saturation modifier should not be used for any of these four instructions. Source and destination operands must be DWord integers. Source and destination must be of the same type, signed integer or unsigned integer. If dst is UD, src0 and src1 may be UD and/or D. However, if any of src0 and src1 is D, source modifier (abs) must be present to convert it to match with dst. If dst is D, src0 and src1 must also be D. They cannot be UD as it may cause unexpected overflow because the computed results are limited to 64 bits.

Format:
[(pred)] mach[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Accumulator is an implicit source and thus cannot be an explicit source operand.

Restriction : AccWrEn is required.

Syntax

```
[(pred)] mach[.cmod] (exec_size) reg reg reg [(pred)] mach[.cmod] (exec_size) reg reg imm32
```

Pseudocode

mach - Multiply Accumulate High

```
Evaluate(WrEn);
   for ( n = 0; n < exec_size; n++ ) {
      if ( WrEn.chan[n] ) {
         acc.chan[n][63:0] = (src1.chan[n][31:16] *
            src0.chan[n][31:0]) « 16 + acc.chan[n][63:0];
         dst.chan[n][31:0] = acc.chan[n][63:32];
      }
   }
```

Errata	Description
	A source modifier must not be used on src1 for the macro operation. This applies to both mul and mach of the macro. If source modifier is required, an additional mov instruction may be used before the macro.

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
D	D
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Multiply Add

mad - Multiply Add

Source: Eulsa

Length Bias: 4

The mad instruction takes component-wise multiplication of src1 and src2, adds the results with the corresponding src0 values, and then stores the final results in dst.
The conditional modifier and saturation (.sat) must not be used when src1 or src2 are dwords.

Format:
[(pred)] mad[.cmod] (exec_size) dst src0 src1 src2

Restriction

Restriction : No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.

Restriction : All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] mad[.cmod] (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn);
for ( n = 0; n < exec_size; n++ ) {
   if ( WrEn.chan[n] ) {
      dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n];
   }
}
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F
DF	DF
HF	HF
HF, F	HF, F

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

mad - Multiply Add

	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group
	48	**Reserved** Format: MBZ
	47	**NibCtrl** Format: NibCtrl
	46	**Reserved** Format: MBZ
	45:44	**Destination Data Type** This field contains the data type for the destination

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

	48:42	**Reserved** Format: MBZ

mad - Multiply Add

	43:42	**Source Data Type** This field contains the data type for all three sources **Value** / **Name** 00b: Single Precision Float 01b: DWord 10b: Unsigned DWord 11b: Double Precision Float
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	34	**Flag Register Number** This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Destination Register File** **Value** / **Name** 0: GRF 1: MRF
	32	**Reserved** Format: MBZ

mad - Multiply Add			
	31:0	**Header**	
		Format:	EU_INSTRUCTION_HEADER

Multiply Add for Macro

madm - Multiply Add for Macro

Source: Eulsa

Length Bias: 4

The madm instruction takes component-wise multiplication of src1 and src2, adds the results with the corresponding src0 values, and then stores the final results in dst.

The source and destination operands have a higher precision carried in the exponent for this operation. The madm instruction is used for macro operations, where precision is accumulated over several instructions. This accumulation requires the exponent to increase by 2 extra bits across multiple madm operations.

Refer to Macros Defined in 'Math' Section for usage and restrictions of this operation.

Format:

[(pred)] madm[.cmod] (exec_size) dst src0 src1 src2

Restriction

Restriction : Accumulator access is restricted to the special accumulators (acc2-acc9). Refer to the Accumulator Section for details on the special accumulators.

Syntax

```
[(pred)] madm[.cmod] (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) {
dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F
DF	DF

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

madm - Multiply Add for Macro

	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are *x*, *y*, *z*, and *w*, respectively, where *x* corresponds to Channel 0 in the group and *w* corresponds to channel 3 in the group
	48	**Reserved** Format: MBZ
	47	**NibCtrl** Format: NibCtrl
	46	**Reserved** Format: MBZ
	45:44	**Destination Data Type** This field contains the data type for the destination

Value	Name
00b	Single Precision Float
01b	DWord
10b	Unsigned DWord
11b	Double Precision Float

	48:42	**Reserved** Format: MBZ
	43:42	**Source Data Type** This field contains the data type for all three sources

Value	Name
00b	Single Precision Float
01b	DWord

madm - Multiply Add for Macro

		10b: Unsigned DWord 11b: Double Precision Float
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	34	**Flag Register Number** This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Destination Register File** Value: 0 — Name: GRF Value: 1 — Name: MRF
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

No Operation

nop - No Operation

Source: Eulsa

Length Bias: 4

Do nothing. The nop instruction takes an instruction dispatch but performs no operation. It can be used for assembly patching in memory, or to insert a delay in the program sequence.

Format:
nop

Restriction

Restriction : The nop instruction takes no instruction options other than Breakpoint.

Syntax

```
nop
```

Pseudocode

```
{ ; // The null statement, which does nothing. }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:31	**Reserved** Format: MBZ
	30	**DebugCtrl** This field allows the insertion of a breakpoint at the current instruction. When the bit is set, hardware automatically stores the current IP in CR register and jumps to the System IP (SIP) BEFORE executing the current instruction. Value 0: No Breakpoint **[Default]** Value 1: Breakpoint
	29:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Oword Aligned Block Read MSD

MSD0R_OWAB - Oword Aligned Block Read MSD

Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: OW Aligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 01h Format: Opcode Aligned Block Read message
	13	**Reserved** Format: MBZ Ignored
	12:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Block Read MSD

MSD0R_OWB - Oword Block Read MSD

Source:	DataPort 0
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 00h Format: Opcode Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Block Write MSD

MSD0W_OWB - Oword Block Write MSD

Source:	DataPort 0
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 08h Format: Opcode Block Write message
	13	**Reserved** Format: MBZ Ignored
	12:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Dual Block Read MSD

MSD0R_OWDB - Oword Dual Block Read MSD

Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 02h Format: Opcode Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:10	**Reserved** Format: MBZ Ignored
	9:8	**Data Elements** Format: MDC_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Dual Block Write MSD

MSD0W_OWDB - Oword Dual Block Write MSD

Source:	DataPort 0
Length Bias:	1
Family:	Block R/W
Group:	OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Ah Format: Opcode Oword Block Read message
	13:10	**Reserved** Format: MBZ Ignored
	9:8	**Data Elements** Format: MDC_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

PIPE_CONTROL

PIPE_CONTROL

Source: RenderCS

Length Bias: 2

The PIPE_CONTROL command is used to effect the synchronization described above.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 2h PIPE_CONTROL Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h PIPE_CONTROL Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 4h DWORD_COUNT_n Format: =n Total Length - 2. Excludes DWord (0,1).
1	31	**Reserved** Format: MBZ
	30	**Reserved** Format: MBZ
	29	**Reserved** Format: MBZ
	28	**Reserved** Format: MBZ
	27	**Reserved**

PIPE_CONTROL

26 **Flush LLC**

Format:	Enable

If enabled, at the end of the current pipe-control the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.

Programming Notes
SW must always program Post-Sync Operation to "Write Immediate Data" when Flush LLC is set.

25 **Reserved**

Format:	MBZ

24 **Destination Address Type**

Defines address space of Destination Address

Value	Name	Description
0h	PPGTT	Use PPGTT address space for DW write
1h	GGTT	Use GGTT address space for DW write

Programming Notes
Ignored if ""No Write" is selected in Operation.

23 **LRI Post Sync Operation**

Value	Name	Description
0h	No LRI Operation	No LRI operation occurs as a result of this instruction. The Post-Sync Operation field is valid and may be used to specify an operation.
1h	MMIO Write Immediate Data	Write the DWord contained in Immediate Data Low (DW3) to the MMIO offset specifed in the Address field.

Programming Notes
This bit caues a post sync operation with an LRI (Load Register Immediate) operation. If this bit is set then the Post-Sync Operation field must be cleared.

22 **Reserved**

21 **Store Data Index**

Format:	U1

Description
Ring Buffer Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). Execlist Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination

PIPE_CONTROL

address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).

20 **Command Streamer Stall Enable**

Format:	U1

If ENABLED, the sync operation will not occur until all previous flush operations pending a completion of those previous flushes will complete, including the flush produced from this command. This enables the command to act similar to the legacy MI_FLUSH command.

19 **Global Snapshot Count Reset**

Format:	U1

Value	Name	Description
0h	Don't Reset	Do not reset the snapshot counts or Statistics Counters.
1h	Reset	Reset the snapshot count in Gen4 for all the units and reset the Statistics Counters except as noted above.

Programming Notes

TIMESTAMP is not reset by PIPE_CONTROL with this bit set.
When Post Sync Operation is set to "Write PS Depth Count" along with Global Snapshot Count Reset, PS Depth Count is Reported first before resetting the value.

18 **TLB Invalidate**

Format:	U1

If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting

If ENABLED, PIPE_CONTROL command will flush the in flight data written out by render engine to Global Observation point on flush done. Also Requires stall bit ([20] of DW1) set.

Programming Notes

If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting.

Post Sync Operation or CS stall must be set to ensure a TLB invalidate occurs. Otherwise no cycle will occur to the TLB cache to invalidate.

17 **Reserved**

Format:	MBZ

PIPE_CONTROL

16 **Generic Media State Clear**

Format:	Disable

If set, all generic media state context information will be invalidated. Any state invalidated will not be saved as part of the render engine context image. The state only only become valid once it is parsed by the command streamer.

15:14 **Post Sync Operation**

Description
This field specifies an optional action to be taken upon completion of the synchronization operation.
This field must be cleared if the LRI Post-Sync Operation bit is set.

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address
2h	Write PS Depth Count	Write the 64-bit PS_DEPTH_COUNT register to the Destination Address
3h	Write Timestamp	Write the 64-bit TIMESTAMP register to the Destination Address

Programming Notes
If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

Workaround
Workaround : PIPECONTROL command with “Command Streamer Stall Enable” must be programmed prior to programming a PIPECONTROL command with Post Sync Op in GPGPU mode of operation (i.e when PIPELINE_SELECT command is set to GPGPU mode of operation).

13 **Depth Stall Enable**

Format:	Enable

This bit must be set when obtaining a "visible pixel" count to preclude the possible inclusion in the PS_DEPTH_COUNT value written to memory of some fraction of pixels from objects initiated after the PIPE_CONTROL command.

Value	Name	Description
0h	Disable	3D pipeline will not stall subsequent primitives at the Depth Test stage.
1h	Enable	3D pipeline will stall any subsequent primitives at the Depth Test stage until the Sync and Post-Sync operations complete.

Programming Notes

PIPE_CONTROL

		This bit must be DISABLED for operations other than writing PS_DEPTH_COUNT. This bit will have no effect (besides preventing write cache flush) if set in a PIPE_CONTROL command issued to the Media pipe.
	12	**Render Target Cache Flush Enable** Format: Enable Setting this bit will force Render Cache to be flushed to memory prior to this synchronization point completing. This bit must be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization. **Value / Name / Description** 0h / Disable Flush / Render Target Cache is NOT flushed. 1h / Enable Flush / Render Target Cache is flushed. **Programming Notes** This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit must not be set when Depth Stall Enable bit is set in this packet.
	11	**Instruction Cache Invalidate Enable** Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 at the top of the pipe i.e. at the parsing time.
	10	**Texture Cache Invalidation Enable** Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the texture caches at the top of the pipe i.e. at the parsing time. **Workaround** Workaround : "CS Stall" bit in PIPE_CONTROL command must be always set for GPGPU workloads when "Texture Cache Invalidation Enable" bit is set
	9	**Indirect State Pointers Disable** Format: Enable **Description** At the completion of the post-sync operation associated with this pipe control packet, the indirect state pointers in the hardware are considered invalid; the indirect pointers are not saved in the context. If any new indirect state commands are executed in the command stream while the pipe control is pending, the new indirect state commands are preserved.

PIPE_CONTROL

		Using Invalidate State Pointer (ISP) only inhibits context restoring of Push Constant (3DSTATE_CONSTANT_*) commands. Push Constant commands are only considered as Indirect State Pointers. Once ISP is issued in a context, SW must initialize by programming push constant commands for all the shaders (at least to zero length) before attempting any rendering operation for the same context.
	8	**Notify Enable** Format: Enable If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.
	7	**Pipe Control Flush Enable** Format: Enable Hardware on parsing PIPECONTROL command with Pipe Control Flush Enable set will wait for all the outstanding post sync operations corresponding to previously executed PIPECONTROL commands are complete before making forward progress.
	6	**Reserved**
	5	**DC Flush Enable** Format: Enable Setting this bit enables flushing of the L3$ portions that caches DC writes. **Programming Notes** DC Flush (L3 Flush) by default doesn't result in flushing/invalidating the IA Coherent lines from L3$, however this can be achieved by setting control bit "**Pipe line flush Coherent lines"** in "L3SQCREG4" register.
	4	**VF Cache Invalidation Enable** Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of VF address based cache at the top of the pipe i.e. at the parsing time. **Programming Notes** If the VF Cache Invalidation Enable is set to a 1 in a PIPE_CONTROL, a separate Null PIPE_CONTROL, all bitfields sets to 0, with the VF Cache Invalidation Enable;set to 0 needs to be sent prior to the PIPE_CONTROL with VF Cache Invalidation Enable set to a 1. As the VF, VFR data caches do not support a full address tag, SW shall invalidate the VF, VFR cache before a draw call that uses modified VERTEX_BUFFER state. Failure to do so may result in erroneous hits in the cache. **Workaround**

PIPE_CONTROL

		Workaround: When VF Cache Invalidate is set "Post Sync Operation" must be enabled to "Write Immediate Data" or "Write PS Depth Count" or "Write Timestamp".
	3	**Constant Cache Invalidation Enable** Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the constant cache at the top of the pipe i.e. at the parsing time.
	2	**State Cache Invalidation Enable** Format: Enable Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 state caches at the top of the pipe i.e. at the parsing time.
	1	**Stall At Pixel Scoreboard** Format: Enable Defines the behavior of PIPE_CONTROL command at the pixel scoreboard.

Value	Name	Description
0h	Disable	Stall at the pixel scoreboard is disabled.
1h	Enable	Stall at the pixel scoreboard is enabled.

Programming Notes
This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit is ignored if Depth Stall Enable is set. Further the render cache is not flushed even if Write Cache Flush Enable bit is set.

	0	**Depth Cache Flush Enable** Format: Enable Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches. This bit applies to HiZ cache, Stencil cache and depth cache.

Value	Name	Description
0h	Flush Disabled	Depth relates caches (HiZ, Stencil and Depth) are NOT flushed.
1h	Flush Enabled	Depth relates caches (HiZ, Stencil and Depth) are flushed.

Programming Notes
Ideally depth caches need to be flushed only when depth is required to be coherent in memory for later use as a texture, source or honoring CPU lock. This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.

<table>
<tr><th colspan="3">PIPE_CONTROL</th></tr>
<tr><td rowspan="2">2</td><td>31:2</td><td>Address
<table><tr><td>Format:</td><td>GraphicsAddress[31:2]U32</td></tr></table>If Post Sync Operation is set to 1h LRI Post-Sync Operation must be clear): Bits 31:3 secify the QW address of where the Immediate Data following this DW in the packet to be stored. Bit 2 MBZ Ignored if "No Write" is the selected in Post-Sync Operation. If LRI Post-Sync Operation is set: Bits 22:2 (Bits 31:23 are reserved MBZ) specify the MMIO offset destination for the data in the Immediate Data Low (DW3) field. Only DW writes are valid.</td></tr>
<tr><td>1:0</td><td>Reserved
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>3</td><td>31:0</td><td>Address High
<table><tr><td>Format:</td><td>GraphicsAddress[63:32]U32</td></tr></table>This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
This field is valid only if the post-sync operation is not 0 and the LRI Post-Sync Operation is clear.</td></tr>
<tr><td>4..5</td><td>63:0</td><td>Immediate Data
<table><tr><td>Format:</td><td>U64</td></tr></table>This field specifies the QWord value to be written to the targeted location.
Only valid when Post-Sync Operation is 1h (Write Immediate Data) or LRI Post-Sync Operation is set.
Ignored if Post-Sync Operation is "No write", "Write PS_DEPTH_COUNT" or "Write TIMESTAMP".</td></tr>
</table>

PIPELINE_SELECT

PIPELINE_SELECT

Source: BSpec

Length Bias: 1

Description

The PIPELINE_SELECT command is used to specify which GPE pipeline is to be considered the 'current' active pipeline. Issuing 3D-pipeline-specific commands when the Media pipeline is selected, or vice versa, is UNDEFINED.

Issuing 3D-pipeline-specific commands when the GPGPU pipeline is selected, or vice versa, is UNDEFINED.

Programming common non pipeline commands (e.g., STATE_BASE_ADDRESS) is allowed in all pipeline modes.

Programming Notes

Software must ensure all the write caches are flushed through a stalling PIPE_CONTROL command followed by another PIPE_CONTROL command to invalidate read only caches prior to programming MI_PIPELINE_SELECT command to change the Pipeline Select Mode.

Example:

... Workload-3Dmode

PIPE_CONTROL (CS Stall, Depth Cache Flush Enable, Render Target Cache Flush Enable, DC Flush Enable)

PIPE_CONTROL (Constant Cache Invalidate, Texture Cache Invalidate, Instruction Cache Invalidate, State Cache invalidate)

PIPELINE_SELECT (GPGPU)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 1h GFXPIPE_SINGLE_DW Format: OpCode
	26:24	**3D Command Opcode** Format: OpCode Value 1h: GFXPIPE_NONPIPELINED **[Default]**
	23:16	**3D Command Sub Opcode** Default Value: 04h PIPELINE_SELECT Format: OpCode

PIPELINE_SELECT

	15:8	**Mask Bits** **Programming Notes** Must be set to modify corresponding bits in Bits 7:0. (For implemented bits)
	7	**Reserved**
	6	**Reserved**
	5	**Force Media Awake** Format: Enable **Value** / **Name** / **Description** 0 / Disabled / Command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. 1 / Enabled / Command streamer sends message to PM to force awake media engine (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [13] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed. Example for usage model: **RCS Ring Buffer**: **PIPELINE_SELECT (Force Media Awake set to '1')** MI_SEMPAHORE_SINGAL (Signal context id 0xABC to Render Command Streamer) **PIPELINE_SELECT (Force Media Awake set to '0')** **MI_BATCH_BUFFER_START** STATE Commands .. .. **PIPELINE_SELECT (Force Media Awake set to '1')** MI_LOAD_REGISTER_IMM (Load register 0x23XX in render command streamer with data 0xFFF) **PIPELINE_SELECT (Force Media Awake set to '0')** ..... **MI_BATCH_BUFFER_END**
	4	**Media Sampler DOP Clock Gate Enable** Format: Enable **Value** / **Name** / **Description** 0 / Disabled / Command Streamer sends message to PM to disable sampler DOP Clock Gating. 1 / Enabled / Command Streamer sends message to PM to enable media sampler DOP Clock Gating.

PIPELINE_SELECT

Programming Notes
Mask bit [12] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.

3 **Render Sampler Power Gate Enable**

Format:	Enable

Value	Name	Description
0	Disabled	Command Streamer sends message to PM to disable render sampler Power Gating.
1	Enabled	Command Streamer sends message to PM to enable render sampler Power Gating.

Programming Notes
Mask bit [11] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.

2 **Render Slice common Power Gate Enable**

Format:	Enable

Value	Name	Description
0	Disabled	Command Streamer sends message to PM to disable render slice common Power Gating.
1	Enabled	Command Streamer sends message to PM to enable render slice common Power Gating.

Programming Notes
Mask bit [10] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.

1:0 **Pipeline Selection**

Value	Name	Description
0	3D	3D pipeline is selected
1	Media	Media pipeline is selected (Includes HD optical disc playback, HD video playback, and generic media workloads)
2	GPGPU	GPGPU pipeline is selected

Programming Notes
Mask bits [9:8] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed. Setting only one of the mask bit [9] or [8] is illegal.

Plane

pln - Plane

Source: Eulsa

Length Bias: 4

The pln instruction computes a component-wise plane equation (w = p*u+q*v+r where u/v/w are vectors and p/q/r are scalars) of src0 and src1 and stores the results in dst. src1 is the input vector u.
src0 provides input scalars p, q, and r, where p is the scalar value based on the region description of src0 and q and r are the scalar values implied from the src0 region. Specifically, q is the second component and r is the fourth component of the 4-tuple (128-bit aligned) that p belongs to.

Format:
[(pred)] pln[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : This is a specialized instruction that only supports an execution size (ExecSize) of 8 or 16.

Restriction : The src0 region must be a replicated scalar (with HorzStride == VertStride == 0).

Restriction : src0 must specify .0 or .4 as the subregister number, corresponding to a subregister byte offset of 0 or 16.

Restriction : If ExecSize is 16 then accumulator cannot be used on src1.

Syntax

```
[(pred)] pln[.cmod] (exec_size) reg reg reg
```

Pseudocode

```
    Evaluate(WrEn);
    for ( n = 0; n < exec_size; n++ ) {
        float dwP = src0.RegNum.SubRegNum[bits4:2];          // A DWord-aligned scalar.
        float dwQ = src0.RegNum.(SubRegNum[bit4:2] | 0x1);   // Second component.
        float dwR = src0.RegNum.(SubRegNum[bit4:2] | 0x3);   // Fourth component.
        if ( ExecSize == 8 ) {
            u = src1.RegNum
            v = src1.(RegNum + 1)
        } else {
            if ( n < 8 ) {
                u = src1.RegNum
                v = src1.(RegNum + 1)
            } else {
                u = src1.(RegNum + 2)
                v = src1.(RegNum + 3)
            }
        }

        if ( WrEn.chan[n] ) {
           dst.chan[n] = dwP * u.chan[n] + dwQ * v.chan[n] + dwR;
        }
    }
```

pln - Plane

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	N

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Read Surface Info MSD

MSD_RSI - Read Surface Info MSD		
Source:	Read-Only DataPort	
Length Bias:	1	
Family:	Other	
Group:	Read Surface Info	
DWord	**Bit**	**Description**
0	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 06h Format: Opcode Read Surface Info message
	13:8	**Reserved** Format: MBZ Ignored
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

REP16 Render Target Write MSD

MSD_RTW_REP16 - REP16 Render Target Write MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTW_REP16 - REP16 Render Target Write MSD

17:14 Message Type

Default Value:	0Ch
Format:	Opcode

Render Target Write message

13 Per-Sample PS Enable

Format:	Enable

If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.

Programming Notes

This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.

When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.

When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).

When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.

12 Last Render Target Select

Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set.

In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 Slot Group Select

Format:	MDC_RT_SGS

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTW_REP16 - REP16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Return

ret - Return

Source: Eulsa

Length Bias: 4

Description

Return execution to the code sequence that called a subroutine.
The ret instruction can be predicated or non-predicated. If non-predicated, all channels jump to the return IP in the first channel of src0 and restore CallMask from the second channel of src0. If predicated, the enabled channels jump to the return IP from the first channel of src0 and the corresponding bits in the CallMask are cleared to zero; if all CallMask bits are zero after the ret instruction, then execution jumps to the return IP from the first channel of src0.
When SPF is on, the predication control must be scalar.

Format:
[(pred)] ret (exec_size) null src0

Restriction

Restriction : This instruction cannot take accumulator as source.

Syntax

```
[(pred)] ret (exec_size) null reg
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { PcIP[n]
= src0.chan[0]; CallMask[n] = 0; } else { PcIP[n] = IP + 1; } } for ( n =
exec_size; n < 32; n++ ) { PcIP[n] = IP + 1; } if ( CallMask[n:0] == 0) ) { //
all channels are zero Jump(src0.chan[0]); CallMask = src0.chan[1]; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types
D, UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32

ret - Return		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Round Down

rndd - Round Down

Source: Eulsa

Length Bias: 4

The rndd instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of src0 and storing the rounded integral float results in dst. This is commonly referred to as the floor() function.
Each result follows the rules in the following tables based on the floating-point mode.

Format:
[(pred)] rndd[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] rndd[.cmod] (exec_size) reg reg [(pred)] rndd[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = floor(src0.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Round to Nearest or Even

rnde - Round to Nearest or Even

Source: Eulsa

Length Bias: 4

The rnde instruction takes component-wise floating point round-to-even operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-even increments stored in the rounding increment bits. The round-to-even increment must be added to the results in dst to create the final round-to-even values to emulate the round-to-even operation, commonly known as the round() function. The final results are the one of the two integral float values that is nearer to the input values. If the neither possibility is nearer, the even alternative is chosen.

Each result follows the rules in the following tables based on the floating-point mode.

Format:

[(pred)] rnde[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] rnde[.cmod] (exec_size) reg reg [(pred)] rnde[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( src0.chan[n] - floor(src0.chan[n]) > 0.5f ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else if ( src0.chan[n] - floor(src0.chan[n]) < 0.5f ) { dst.chan[n] = floor(src0.chan[n]); } else { if ( floor(src0.chan[n]) is odd ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = floor(src0.chan[n]); } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS

rnde - Round to Nearest or Even		
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Round to Zero

rndz - Round to Zero

Source: Eulsa

Length Bias: 4

The rndz instruction takes component-wise floating point round-to-zero operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in dst to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the truncate() function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Format:
[(pred)] rndz[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] rndz[.cmod] (exec_size) reg reg [(pred)] rndz[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = floor(src0.chan[n]); if ( abs(src0.chan[n]) < abs(dst.chan[n]) ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = floor(src0.chan[n]); } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Round Up

rndu - Round Up

Source: Eulsa

Length Bias: 4

The rndu instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of src0, commonly known as the ceiling() function.
Each result follows the rules in the following tables based on the floating-point mode.

Format:
[(pred)] rndu[.cmod] (exec_size) dst src0

Syntax

```
[(pred)] rndu[.cmod] (exec_size) reg reg [(pred)] rndu[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( src0.chan[n] - floor(src0.chan[n]) > 0.0f ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = src0.chan[n]; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Sampler Cache Media Block Read MSD

MSD_SC_MB - Sampler Cache Media Block Read MSD

Source: Read-Only DataPort
Length Bias: 1
Family: Other
Group: Media Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 05h Format: Opcode Media Block Read Sampler Cache message
	13:11	**Reserved** Format: MBZ Ignored
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Sampler Cache Oword Unaligned Block Read MSD

MSD_SC_OWUB - Sampler Cache Oword Unaligned Block Read MSD

Source:	Read-Only DataPort
Length Bias:	1
Family:	Block R/W
Group:	OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 04h Format: Opcode Oword Unaligned Block Read Sampler Cache message
	13:11	**Reserved** Format: MBZ Ignored
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Scaled Untyped Surface Read MSD

MSD2R_US - Scaled Untyped Surface Read MSD

Source:	DataPort 2
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scaled Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A32_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 01h Format: Opcode Untyped Surface Read message
	14	**Reserved** Format: MBZ Ignored
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Sideband Scaled Offset** Format: MDC_A32_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

Scaled Untyped Surface Write MSD

MSD2W_US - Scaled Untyped Surface Write MSD

Source:	DataPort 2
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scaled Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_A32_MHP If present, modifies the address calculations.
	18:15	**Message Type** Default Value: 09h Format: Opcode Untyped Surface Write message
	14	**Reserved** Format: MBZ Ignored
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Sideband Scaled Offset** Format: MDC_A32_SBSO In combination with Header Present field, specifies the Scale pitch and the Offset for the message.

Scattered Move

smov - Scattered Move

Source: Eulsa

Length Bias: 4

The smov instruction moves the components in src0 into dst.
For each enabled channel, copy src0 to dst. The immediate is used to selectively enable channels without using flags.
When predication is enabled, the predicate mask is not generated from the flags. Instead, the immediate is used to mask the execution mask. If any channel is enabled as a result of this masking, the instruction is executed.
When predication is not enabled, the immediate masks the execution mask. This provides flexibility to mask out any channel with an immediate.

Format:
[(pred)] smov[.cmod] (exec_size) dst src0 src1

Programming Notes

When predication is disabled, the immediate provides the flexibility to perform a select operation without the use of flags.

When predication is enabled, the usage model provides flexibility to select any bit in the flag registers for predication for execution size of 1.

Syntax

```
[(pred)] smov[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
if pred emask = OR (emask AND imm32) Else pmask = imm32. Evaluate(WrEn); for ( n
= 0; n < 32; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
*W,*D, HF, F	*W,*D, HF, F
*W,*D, HF, F	*Q, DF
*Q, DF	*W,*D, HF, F
*Q, DF	*Q, DF

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

smov - Scattered Move		
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Scratch Block Read MSD

MSD0R_SB - Scratch Block Read MSD

Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 0h Format: Opcode Scratch Block Read message
	16	**Channel Mode** Format: MDC_CMODE Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	15	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	14	**Reserved** Format: MBZ Ignored
	13:12	**Data Elements** Format: MDC_DB_HW Specifies the number of registers to be read or written

MSD0R_SB - Scratch Block Read MSD		
	11:0	**Address Offset** Format: GeneralStateOffset[17:6] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Scratch Block Write MSD

MSD0W_SB - Scratch Block Write MSD

Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 1h Format: Opcode Scratch Block Write message
	16	**Channel Mode** Format: MDC_CMODE Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	15:14	**Reserved** Format: MBZ Ignored
	13:12	**Data Elements** Format: MDC_DB_HW Specifies the number of registers to be read or written
	11:0	**Address Offset** Format: GeneralStateOffset[17:6] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Select

sel - Select

Source: Eulsa

Length Bias: 4

Description

The sel instruction selectively moves the components in src0 or src1 into the channels of dst based on the predication. On a channel by channel basis, if the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst.

As the predication is used to select the two sources, it is not included in the evaluation of WrEn. The predicate clause is mandatory if cmod is omitted/0000b. If both predication and the conditional modifier are omitted, the results are undefined.

If the conditional modifier is specified (not 0000b, a compare is performed and the resulting condition flag is used for the sel instruction. Conditional modifiers .ge and .l follow the cmpn rules, and all other conditional modifiers follow the cmp rules. Predication is not allowed in this mode.

A sel instruction with cmod .l is used to emulate a MIN instruction.

A sel instruction with cmod .ge is used to emulate a MAX instruction.

For a sel instruction with a .l or .ge conditional modifier, if one source is NaN and the other not NaN, the non-NaN source is the result. If both sources are NaNs, the result is NaN. For all other conditional modifiers, if either source is NaN then src1 is selected.

A sel instruction without a conditional modifier always copies a denorm source value to a denorm destination value (in the manner of a raw move). This applies even if the source modifies are set on the *sel* instruction sources.

The sel instruction uses any conditional modifier internally and does not update the flag register if a conditional modifier is used.

A sel instruction with cmod or source modifier will flush denorm to zero, depending on the denorm mode bit; a sel instruction without cmod and source modifier will retain denorm.

Format:
(pred) sel[.cmod] (exec_size) dst src0 src1

Syntax

```
(pred) sel[.cmod] (exec size) reg reg reg (pred) sel[.cmod] (exec size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn, NoPMask); if (cmod == "0000") { // no CMod Evaluate(PMask); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( PMask.channel[n] ) { dst.chan[n] = src0.chan[n]; } else { dst.chan[n] = src1.chan[n]; } } } } else { // with CMod Evaluate(CMod); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( CMod.chan[n] ) { dst.chan[n] = src0.chan[n]; } else { dst.chan[n] = src1.chan[n]; } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier

sel - Select

Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
F	F
DF	DF
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q
HF	HF
HF, F	HF, F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_REG**
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_IMM**
	63:32	**Operand Controls** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Send Message

send - Send Message

Source: Eulsa

Length Bias: 4

Description
Send a message stored in GRF starting at <src> to a shared function identified by <ex_desc> along with control from <desc> with a GRF writeback location at <dest>.
The send instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The send instruction adds an entry to the EU's message request queue. The request message is stored in a block of contiguous GRF registers. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src> is the lead GRF register for request. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend mesage descriptor field <ex_desc> contains the target function ID. WrEn is forwarded to the target function in the message sideband.
The extended message descriptor field <ex_desc> also contains the extended function control field to be sent to the Target Shared Function over message sideband.
The send instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function. Message descriptor field <desc> can be a 32-bit immediate, imm32, or a 32-bit scalar register, <reg32a>. GEN restricts that the 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0<0;1,0>:ud. When <desc> is a register operand, only the lower 29 bits of <reg32a> are used.
<ex_desc> is a 32-bit immediate, imm32. The lower 4bits of the <ex_desc> specifies the SFID for the message. The bit5 of the extended message descriptor, the EOT field, always comes from bit 127 of the instruction word. A thread must terminate with a send instruction with EOT turned on. The higher 16bits, bit31:16 specify the 16bit extended function control field. Interpretation of the extended function control signals is subject to the target external function.
<src> is a 256-bit aligned GRF register. It serves as the leading GRF register of the request.
The source dependency control, {NoSrcDepSet} is used to control the setting of source dependency for the source.
<dest> serves for two purposes: to provide the leading GRF register location for the response message if present, and to provide parameters to form the channel enable sideband signals. <dest> signals whether there is a response to the message request. It can be either a null register, a direct-addressed GRF register or a register-indirect GRF register. Otherwise, hardware behavior is undefined. If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null. If <dest> is a GRF register, the register number is forwarded to the

send - Send Message

shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off. The subregister number, horizontal stride, destination mask and type fields of <dest> are always valid and are used in part to generate on the WrEn. This is true even if <dest> is a null register (this is an exception for null as for most cases these fields are ignored by hardware). The 16-bit channel enables of the message sideband are formed based on the WrEn. Interpretation of the channel enable sideband signals is subject to the target external function. In general for a 'send' instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases. The destination dependency control, {NoDDClr}, can be used in this instruction. This allows software to control the destination dependencies for multiple 'read'-type messages similar to that for multiple instructions using EU execution pipeline. As send does not check register dependencies for the post destination, {NoDDChk} should not be used for this instruction.

Restriction

Restriction : Software must obey the following rules in signaling the end of thread using the send instruction: The posted destination operand must be null. No acknowledgement is allowed for the send instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource. A thread must terminate with a send instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a send instruction with message to the following shared functions: Sampler unit, NULL function For example, a thread may terminate with a URB write message or a render cache write message. A root thread originated from the media (generic) pipeline must terminate with a send instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed description. The send instruction can not update accumulator registers. Saturate is not supported for send instruction. ThreadCtrl encodings Switch is not supported for send instruction. The send with EOT should use register space R112-R127 for <src>. This is to enable loading of a new thread into the same slot while the message with EOT for current thread is pending dispatch. Any instruction updating the ARF must use a {Switch} if the ARF is not used before EOT. DepCtrl Must not be used with Send Instruction. When pagefault is enabled, the source and destination operands must not overlap. This is required to ensure the messages can be replayed.</src>

Restriction :
The source dependency control, {NoSrcDepSet}, must not be set for the send instruction preceding a send instruction with EOT.

Syntax

```
[(pred)] send (exec_size) reg reg imm32 reg32a [(pred)] send (exec_size) reg reg imm32 imm32
```

Pseudocode

```
Evaluate(WrEn); <MsgChEnable> = WrEn; <SourceReg> = <src>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg>, <desc>, <ex_dest>);
```

Predication	Conditional Modifier	Saturation	Source Modifier

send - Send Message

Y	N	N	N

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95:89	**Flags** Format: EU_INSTRUCTION_FLAGS
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Shared Function ID (SFID)** Format: SFID
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

SFC_AVS_CHROMA_Coeff_Table

SFC_AVS_CHROMA_Coeff_Table

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 5h SFC_AVS CHROMA Coeff_Table Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 7Fh Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:56	**Table 1Y Filter Coefficient[[n],5]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for Y-direction.

SFC_AVS_CHROMA_Coeff_Table

		Programming Notes For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	55:48	**Table 1X Filter Coefficient[[n],5]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	47:40	**Table 1Y Filter Coefficient[[n],4]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	39:32	**Table 1X Filter Coefficient[[n],4]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	31:24	**Table 1Y Filter Coefficient[[n],3]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	23:16	**Table 1X Filter Coefficient[[n],3]** Format: S1.6 2's Complement

SFC_AVS_CHROMA_Coeff_Table

		Range: [-2, +2) Chroma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	15:8	**Table 1Y Filter Coefficient[[n],2]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	7:0	**Table 1X Filter Coefficient[[n],2]** Format: S1.6 2's Complement **Range:** [-2, +2) Chroma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
3..64	1983:0	**Filter Coefficients** Format: Chroma_Filter_Coefficients_Array[31]

SFC_AVS_LUMA_Coeff_Table

SFC_AVS_LUMA_Coeff_Table

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 6h SFC_AVS LUMA Coeff_Table Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 7Fh — Name: Excludes DWord (0,1) **[Default]**
1..4	127:120	**Table 0Y Filter Coefficient[[n],7]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction.

SFC_AVS_LUMA_Coeff_Table

		Programming Notes For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	119:112	**Table 0X Filter Coefficient[[n],7]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	111:104	**Table 0Y Filter Coefficient[[n],6]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	103:96	**Table 0X Filter Coefficient[[n],6]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	95:88	**Table 0Y Filter Coefficient[[n],5]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	87:80	**Table 0X Filter Coefficient[[n],5]** Format: S1.6 2's Complement

SFC_AVS_LUMA_Coeff_Table

		Range: [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	79:72	**Table 0Y Filter Coefficient[[n],4]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	71:64	**Table 0X Filter Coefficient[[n],4]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	63:56	**Table 0Y Filter Coefficient[[n],3]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	55:48	**Table 0X Filter Coefficient[[n],3]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	47:40	**Table 0Y Filter Coefficient[[n],2]**

SFC_AVS_LUMA_Coeff_Table

		Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	39:32	**Table 0X Filter Coefficient[[n],2]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	31:24	**Table 0Y Filter Coefficient[[n],1]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	23:16	**Table 0X Filter Coefficient[[n],1]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	15:8	**Table 0Y Filter Coefficient[[n],0]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for Y-direction. **Programming Notes**

SFC_AVS_LUMA_Coeff_Table

		For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
	7:0	**Table 0X Filter Coefficient[[n],0]** Format: S1.6 2's Complement **Range:** [-2, +2) Luma table for X-direction. **Programming Notes** For 5x5 filter mode, table 0, 6 and 7 must be programmed to zero.
5..128	3967:0	**Filter Coefficients** Format: Luma_Filter_Coefficients_Array[31]

SFC_AVS_STATE

SFC_AVS_STATE

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Ah — Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 2h SFC_AVS_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:24	**Sharpness Level** Format: U8 When adaptive scaling is off, determines the balance between sharp and smooth scalers. 0 — — Contribute 1 from the smooth scalar

SFC_AVS_STATE

		255 | Contribute 1 from the sharp scalar
	23:7	**Reserved** Format: MBZ
	6:4	**Transition Area with 4 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 4 pixel calculation.
	3	**Reserved** Format: MBZ
	2:0	**Transition Area with 8 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 8 pixel calculation.
2	31:24	**Reserved** Format: MBZ
	23:16	**Max Derivative 4 Pixels** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 4 pixel area.
	15:8	**Reserved** Format: MBZ
	7:0	**MAX Derivative Point 8** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 8 pixel area.

SFC_FRAME_START

SFC_FRAME_START

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 4h SFC_FRAME_START Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Reserved** Format: MBZ

SFC_IEF_STATE

SFC_IEF_STATE

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h SFC_IEF_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 16h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:28	**Reserved** Format: MBZ
	27:23	**R3c Coefficient** Default Value: 5 Format: U0.5 IEF smoothing coefficient, *see IEF map*.

SFC_IEF_STATE

	22:18	**R3x Coefficient** Default Value: 5 Format: U0.5 IEF smoothing coefficient, *see IEF map*.
	17:12	**Strong Edge Threshold** Default Value: 8 Format: U6 If EM > Strong Edge Threshold → the basic VSA detects a strong edge.
	11:6	**Weak Edge Threshold** Default Value: 1 Format: U6 If Strong Edge Threshold > EM > Weak Edge Threshold → the basic VSA detects a weak edge.
	5:0	**Gain Factor** Default Value: 44 Format: U6 User control sharpening strength.
2	31:27	**R5c Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, *see IEF map*.
	26:22	**R5cx Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, *see IEF map*.
	21:17	**R5x Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, *see IEF map*.
	16:14	**Strong Edge Weight** Default Value: 7 Format: U3 Sharpening strength when a STRONG edge is found in basic VSA.

SFC_IEF_STATE

	13:11	**Regular Weight** Default Value: 2 Format: U3 Sharpening strength when a WEAK edge is found in basic VSA.
	10:8	**Non Edge Weight** Default Value: 1 Format: U3 . Sharpening strength when NO EDGE is found in basic VSA.
	7:0	**Global Noise Estimation** Default Value: 255 Format: U8 Global noise estimation of previous frame.
3	31:28	**Reserved** Format: MBZ
	27:22	**Hue_Max** Default Value: 14 Format: U6 Rectangle half width.
	21:16	**Sat_Max** Default Value: 31 Format: U6 Rectangle half length.
	15:8	**STD Cos(alpha)** Format: S0.7 2's Complement **Default Value** = 79/128
	7:0	**STD Sin(alpha)** Format: S0.7 2's Complement **Default Value** = 101/128
4	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate.

SFC_IEF_STATE

	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate.
	15	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	14:12	**Diamond Margin** Default Value: 4 Format: U3
	11	**Reserved** Format: MBZ
	10:0	**S3U** Format: S2.8 -2's Complement Slope 3 of the upper part of the detection PWLF. **Deafult:** 0/256
5	31	**Skin Detail Factor** Format: U1 Enumerated Type This flag bit is in operation only when one of the following conditions exists: • when the control bit **SkinToneTunedIEF_Enable** is on. • When **SkinDetailFactor** is equal to 0, sign(**SkinDetailFactor**) is equal to +1, and the content of the detected skin tone area is detail revealed. • When **SkinDetailFactor** is equal to 1, sign(**SkinDetailFactor**) is equal to -1, and the content of the detected skin tone area is not detail revealed. **Value** / **Name** 0 – Detail Revealed **[Default]** 1 – Not Detail Revealed
	30:24	**Diamond_du** Default Value: 0 Format: S6 -2's Complement Rhombus center shift in the sat-direction, relative to the rectangle center.

SFC_IEF_STATE

	23:21	**HS_margin** Default Value: 3 Format: U3 Defines rectangle margin.
	20:13	**Diamond_alpha** Format: U2.6 1 / tan(β) **Deafult:** 100/64
	12:7	**Diamond_Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.
	6:0	**Diamond_dv** Default Value: 0 Format: S6 -2's Complement Rhombus center shift in the hue-direction, relative to the rectangle center.
6	31:24	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function.
	23:16	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	15:8	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.

SFC_IEF_STATE

	7:0	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
7	31:16	**Reserved** Format: MBZ
	15:0	**INV_Margin_VYL** Format: U0.16 1 / Margin_VYL **Default:** 3300/65536
8	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**INV_Margin_VYU** Format: U0.16 1 / Margin_VYL **Default:** 1600/65536
9	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.

SFC_IEF_STATE

	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y Point 2 of the lower part of the detection PWLF.
10	31:27	**Y_Slope_2** Format: U2.3 Slope between points Y3 and Y4. **Default:** 31/8
	26:16	**S0L** Format: S2.8 -2's Complement Slope 0 of the lower part of the detection PWLF. **Default:** -5/256
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8 V Bias 2 of the lower part of the detection PWLF.
11	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Format: S2.8 -2's Complement **Default:** 0/256 Slope 2 of the lower part of the detection PWLF.

SFC_IEF_STATE

	10:0	**S1L** Format: S2.8 -2's Complement **Default:** 0/256 Slope 1 of the lower part of the detection PWLF.
12	31:27	**Y_Slope1** Format: U2.3 Slope between points Y1 and Y2. **Default:** 31/8
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Format: S2.8 -2's Complement Slope 3 of the lower part of the detection PWLF. **Default:** 0/256
13	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.

SFC_IEF_STATE

	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
14	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Format: S2.8 -2's Complement Slope 0 of the upper part of the detection PWLF. **Default:** 256/256
	15:8	**B3U** Default Value: 140 Format: U8 V Bias 3 of the upper part of the detection PWLF.
	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
15	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Format: S2.8 -2's Complement **Default:** -179/256 Slope 2 of the upper part of the detection PWLF.

SFC_IEF_STATE

	10:0	**S1U** Format: S2.8 -2's Complement **Default:** 113/256 Slope 1 of the upper part of the detection PWLF.
16	31:29	**Reserved** Format: MBZ
	28:16	**C1** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
	15:3	**C0** Default Value: 1024 Format: S2.10 -2's Complement Transform coefficient
	2	**Reserved** Format: MBZ
	1	**YUV Channel Swap**
	0	**Transform Enable**
17	31:26	**Reserved** Format: MBZ
	25:13	**C3** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
	12:0	**C2** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
18	31:26	**Reserved** Format: MBZ

SFC_IEF_STATE

	25:13	**C5** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
	12:0	**C4** Default Value: 1024 Format: S2.10 -2's Complement Transform coefficient
19	31:26	**Reserved** Format: MBZ
	25:13	**C7** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
	12:0	**C6** Default Value: 0 Format: S2.10 -2's Complement Transform coefficient
20	31:13	**Reserved** Format: MBZ
	12:0	**C8** Default Value: 1024 Format: S2.10 -2's Complement Transform coefficient
21	31:22	**Reserved** Format: MBZ
	21:11	**Offset out 1** Default Value: 0 Format: S2.8 -2's Complement Offset out for Y/R.

SFC_IEF_STATE

	10:0	**Offset in 1** Default Value: 0 Format: S2.8 -2's Complement Offset in for Y/R.
22	31:22	**Reserved** Format: MBZ
	21:11	**Offset out 2** Default Value: 0 Format: S2.8 -2's Complement Offset out for U/G.
	10:0	**Offset in 2** Default Value: 0 Format: S2.8 -2's Complement Offset in for U/G.
23	31:22	**Reserved** Format: MBZ
	21:11	**Offset out 3** Default Value: 0 Format: S2.8 -2's Complement Offset out for V/B.
	10:0	**Offset in 3** Default Value: 0 Format: S2.8 -2's Complement Offset in for V/B.

SFC_LOCK

SFC_LOCK

Source: BSpec

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 0h SFC Lock Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:2	**Reserved** Format: MBZ
	1	**Pre-Scaled Output Surface Output Enable** **VD - Reconstructed Pixel Output Enable** For VD Mode, this field specifies the enabling of writing out the display reconstructed pixel to memory. It could be pre or post- ILDB filter pixel output based on the pre- and post- filter setting in the AVC state command.

SFC_LOCK

Pre-Deblock Flag	Post-Deblock Flag	VD Pixels Output to Memory	VD Pixels Output to SFC
0	0	Invalid for SFC Mode	Invalid for SFC Mode
0	1	Filtered Pixels (allow ON/OFF)	Filter Pixels Sent to SFC for Scaling
1	0	Non-filter (bypass) pixels (allow ON/OFF)	Non-Filter Pixels Sent to SFC for Scaling
1	1	Non-filter (bypass) pixels (allow ON/OFF) Filtered pixels (always OFF)	Filter Pixels Sent to SFC for Scaling.

VE - image enhanced pixel Output Enable

For VE Mode, this field indicates if the VEBOX will enable writing out the image enhanced pixels to memory which is streamed to SFC pipeline for scaling.Filtered data is streamed directly from VEBOX to SFC through a dedicated internal interface. The pixel data send from VE to SFC is YUV format in 12-bit precision irrespective of VEBOX input surface type, pixel precision, chroma format, and color format (RGBA/YUVA). The following table shows allowed usage with VE -image enhanced pixel output enable along with SFC being enabled.

VE Output Surface Format	Bits per channel	Can SFC be enabled ?
Y8/ NV12/ AYUV/ YUYV/YVYU/UYVY/VYUY	8bit	Yes
RGBA8	8bit	No
RGBA10	10bit	No
RGBA16	16bit	No
Y16/ P216/P016/ Y216/ Y416	16bit	Yes

When DN is enabled, DN output is always on.

When DI is Enabled, VE must send the first DI constructed surface to SFC in case VEBOX state indicate two DI frames output, while stream out the second DI constructed surface to memory.

Else the DI output which is enabled will be sent out to SFC. VE output surface Type is programmed in VE_State command.

0	**VE-SFC Pipe Select**

SFC_STATE

SFC_STATE

Source: BSpec

Length Bias: 2

Description
This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value: Ah — Name: Media Misc **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 1h SFC_State Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 1Eh — Name: Excludes DWord (0,1) **[Default]**
1	31:11	**Reserved** Format: MBZ

SFC_STATE

10:8 **VD/VE Input Ordering Mode**

Format:	U3

- VD mode: (SFC pipe mode set as "0")
- VE mode: (pipe mode set as "1 and 4")

For values for each mode, please refer to the table below:

Value	Name	Description	Exists If
0		16x16 block z-scan order - no shift	//VD Mode
1		16x16 block z-scan order - 4 pixels shift upward	//VD Mode
2		8x8 block jpeg z-scan order	//VD Mode
3		16x16 block jpeg z-scan order	//VD Mode
4		16x16 block VP8 row-scan order - no shift	//VD Mode
5-7		Reserved	//VD Mode
0		8x4 block column order, 64 pixel column	//VE Mode
1		4x4 block column order, 64 pixel column	//VE Mode
4-7		Reserved	//VE Mode

Programming Notes

This field shall be programmed according to video modes used in VDBOX. NOTE: SFC supports progressive input and output only (Interlaced/MBAFF is not supported).

Video Mode	Surface Format	SFC Input Chroma Sub-Sampling	VD/VE Input Ordering Mode
AVC w/o LF	Monochrome	0	0
AVC w/o LF	420 (NV12)	1	0
AVC with LF	Monochrome	0	1
VP8 w/o LF	420 (NV12)	1	4
JPEG (YUV Interleaved)	Monochrome	0	2
JPEG (YUV Interleaved)	420	1	3
JPEG (YUV Interleaved)	422H_2Y	2	2
JPEG (YUV Interleaved)	422H_4Y	2	3
JPEG (YUV Interleaved)	444	4	2

This field shall be programmed according to Image enhancement modes used in VEBOX.

VEBOX MODE	VEBOX Single Pipe Enable Bit	SFC Input Surface Format	SFC Input Chroma Sub Sampling	VD/VE Input Ordering Mode
1. DN/HP with RGB input 2. Camera pipe (DM) enabled 3. IECP with FECSC, CCM, FGC filters enabled	1	Monochrome	0	1
	1	420 (NV12)	1	1
	1	422H	2	1
	1	444	4	1
All other modes: (Legacy DN/DI/IECP features)	0	Monochrome	0	0
	0	420 (NV12)	1	0
	0	422H	2	0
	0	444	4	0

SFC_STATE

7:4 **SFC Input Chroma Sub-Sampling**

Value	Name	Description
0	4:0:0	SFC to insert UV channels
1	4:2:0	
2	4:2:2 Horizonatal	VD: 2:1:1
3	Reserved	
4	4:4:4 Progressive/Interleaved	
5-6	Reserved	

Programming Notes

This field shall be programmed according to video modes used in VDBOX. NOTE: SFC supports progressive input and output only (Interlaced/MBAFF is not supported).

Video Mode	Surface Format	SFC Input Chroma Sub-Sampling	VD/VE Input Ordering Mode
AVC w/o LF	Monochrome	0	0
AVC w/o LF	420 (NV12)	1	0
AVC with LF	Monochrome	0	1
VP8 w/o LF	420 (NV12)	1	4
JPEG (YUV Interleaved)	Monochrome	0	2
JPEG (YUV Interleaved)	420	1	3
JPEG (YUV Interleaved)	422H_2Y	2	2
JPEG (YUV Interleaved)	422H_4Y	2	3
JPEG (YUV Interleaved)	444	4	2

This field shall be programmed according to Image enhancement modes used in VEBOX.

VEBOX MODE	Surface Format	SFC Input Chroma Sub Sampling	VD/VE Input Ordering Mode
Legacy DN/DI/IECP features	Monochrome	0	0
Legacy DN/DI/IECP features	420 (NV12)	1	0
Legacy DN/DI/IECP features	422H	2	0
Legacy DN/DI/IECP features	444	4	0
Capture/Camera pipe enabled(Demosaic)	Monochrome	0	1
Capture/Camera pipe enabled(Demosaic)	420 (NV12)	1	1
Capture/Camera pipe enabled(Demosaic)	422H	2	1
Capture/Camera pipe enabled(Demosaic)	444	4	1

SFC_STATE

3:0 SFC Pipe Mode

Value	Name	Description
0		VD-to-SFC AVS
1		VE-to-SFC AVS + IEF + Rotation
2-3		Reserved
4		VE-to-SFC Integral Image
5-15		Reserved

Programming Notes

Note: for SFC Pipe mode set to VE-to-SFC AVS mode.
IECP pipeline mode MUST be enabled.
However, each sub-IECP feature can be turned on/off independently.

2 31:28 Reserved

Format:	MBZ

27:16 Input Frame Resolution Height

Format:	U12-1

Minus 1 in unit of pixel [11:0]. It is set to the value of the output resolution or number of pixels streaming into SFC from VD or VEBox. Since the max value support is 4k pixels, the max value allowed is 4K minus 1.

- VDBOX frame height is multiple of 16 for Video source and JPEG formats other than 400, 444 and 422H_2Y.
- VDBOX frame height is multiple of 8 for JPEG formats 400, 444 and 422H_2Y.
- VEBOX frame height is multiple of 4.

Min Resolution is 128 pixels. *Max Resolution* is upto 4K pixels. e.g. for 1920x1080 content, FrameHeightInMBsMinus1 is equal to1087 (1080 rounded up 16 pixel boundary, minus 1. i.e. effectively specified as 1088 instead).

Restriction

Restriction : For Integral Image Mode, this field is Reserved and MBZ.

15:12 Reserved

Format:	MBZ

SFC_STATE

11:0 Input Frame Resolution Width

Format:	U12-1

Minus 1 in unit of pixel [11:0]. It is set to the value of the output resolution or number of pixels streaming into SFC from VD or VEBox. Since the max value support is 4k pixels, the max value allowed is 4K minus 1.

- VDBOX frame width is multiple of 16 for Video source and JPEG formats other than 400, 444 and 422H_2Y.
- VDBOX frame width is multiple of 8 for JPEG formats 400, 444 and 422H_2Y.
- VEBOX frame width is multiple of 16.

Min Resolution is 128 pixels.
Max Resolution is upto 4K pixels.
e.g. for 1920x1080 content, FrameHeightInMBsMinus1 is equal to1087 (1080 rounded up 16 pixel boundary, minus 1. i.e. effectively specified as 1088 instead).

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.

3 31:17 Reserved

Format:	MBZ

16 Reserved

15 Reserved

14:12 Pre-AVS Chroma Downsampling co-siting position Horizontal Direction

Format:	U3

Value	Name	Description
000b	0/8 (Left full pixel)	0 (fraction_in_integer)
001b	1/8	1 (fraction_in_integer)
010b	1/4 (2/8)	2 (fraction_in_integer)
011b	3/8	3 (fraction_in_integer)
100b	1/2 (4/8)	4 (fraction_in_integer)
101b	5/8	5 (fraction_in_integer)
110b	3/4 (6/8)	6 (fraction_in_integer)
111b	7/8	7 (fraction_in_integer)

11 Reserved

Format:	MBZ

SFC_STATE

10:8 **Pre-AVS Chroma Downsampling co-siting position Vertical Direction**

Format:	U3

This field specifies the fractional position of the bilinear filter for chroma downsampling. In the Y-axis.

Value	Name	Description
000b	0/8 (Left full pixel)	0 (fraction_in_integer)
001b	1/8	1 (fraction_in_integer)
010b	1/4 (2/8)	2 (fraction_in_integer)
011b	3/8	3 (fraction_in_integer)
100b	1/2 (4/8)	4 (fraction_in_integer)
101b	5/8	5 (fraction_in_integer)
110b	3/4 (6/8)	6 (fraction_in_integer)
111b	7/8	7 (fraction_in_integer)

7:6 **Pre- AVS Chroma Downsampling Enable**

Format:	U2

This is set to enable chroma downsample bilinear filtering prior to AVS scaling operation. This should be set only if the input chroma format has higher sampling than the chroma format used in the AVS filtering.

Value	Name
00b	Disable
01b	444 -> 420
10b	444 -> 422
11b	422 -> 420

Programming Notes
For Integral Image Mode, this field is Reserved and MBZ. The programming of this field should match the input format of VEBox.

5 **RGBA_Channel_Swap Enable**

Default Value:	0
Format:	Enable

This bit should only be used with RGB output formats and CSC conversion is turned on. When this bit is set, the R and B channels are swapped into the output RGB channels as shown in the following table:

Name	Bits	MSB Color Order	Swapped
RGBA8	8:8:8:8	A:B:G:R	A:R:G:B
RGBA10	2:10:10:10	A:R:G:B	A:B:G:R
RGB 5:6:5	5:6:5	R:G:B	B:G:R

SFC_STATE

4 — Reserved

Format:	MBZ

3:0 — Output Surface Format type

SFC output surface format type.
Reserved

Value	Name	Description	Exists If
0		AYUV 4:4:4 (8:8:8:8 MSB-A:Y:U:V)	//Tile-Y/ Tile-X/Linear
1		RGBA8 4:4:4:4 (8:8:8:8 MSB-A:B:G:R)	//Tile-Y/ Tile-X/Linear
2		RGBA10 10:10:10:2 (2:10:10:10 MSB-A:R:G:B)	//Tile-Y/ Tile-X/Linear
3		RGB 5:6:5 (5:6:5 MSB-R:G:B)	//Tile-Y/ Tile-X/Linear
4		Planar NV12 4:2:0 8-bit	//Tile-Y
5		Packed YUYV 4:2:2 8-bit	//Tile-Y/ Tile-X/Linear
6		Packed UYVY 4:2:2 8-bit	//Tile-Y/ Tile-X/Linear
7		Packed integral Image 32-bit	//Linear
8		Packed integral Image 64-bit	//Linear

Restriction

Restriction : For Integral Image Mode, output surface format type must be set to 32/64-bit Integral Image Plane.
Driver/SW must ensure the max accumulated integral image value does not exceed the programmable output precision.
HW will simply generate wrong value once it overflow in wrap around case.

4 — 31:22 — Reserved

Format:	MBZ

21:20 — Reserved

Format:	MBZ

19 — CSC Enable

Description

This field specifies the scaled pixels are converted from YUV to RGB. The output format type must be programmed in one of the RGBAx type.

Restriction

Restriction : For Integral Image Mode, this field is Reserved and MBZ.

18 — Color Fill Enable

Programming Notes

This field could be enabled only if the scaled resolution is smaller than the output/display resolution.

SFC_STATE

If enabled, HW will fill the gap with programmable pixel values. Else, nothing will be filled in the gap region.

Usage: Color fill must be enabled for the first time/pass when a new surface is allocated/ used. Optional for subsequence frames since the gap region is filled with default pixels by prior passes.

17:16 **Rotation Mode**

Format:	U2

Value	Name
00b	0 (degrees)
01b	90 Clockwise
10b	180 Clockwise
11b	270 Clockwise

Programming Notes

SFC rotation (90, 180 and 270) should be set only on VEBox input mode and SFC output set to TileY.
Restriction:

- For Integral Image Mode, this field is Reserved and MBZ.
- For VDBox Mode, this field is Reserved and MBZ.
- For linear or TileX SFC output, this field is Reserved and MBZ.

15:13 **Reserved**

Format:	MBZ

12 **Chroma Upsampling Enable**

This field enables the high-quality UV channel upsampler prior to IEF filter process.
This field should be disabled when the source pixels and output pixels are kept with the same chroma sub-sample type and IEF is disabled.

Restriction

Restriction : For Integral Image Mode, this field is Reserved and MBZ.

11:10 **Reserved**

Format:	MBZ

9 **Bypass X Adaptive Filtering**

Value	Name	Description
0	Enable X Adaptive Filtering	
1	Disable X Adaptive Filtering	The X direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value.

SFC_STATE

8 Bypass Y Adaptive Filtering

Value	Name	Description
0	Enable Y Adaptive Filtering	
1	Disable Y Adaptive Filtering	The Y direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value.

7 AVS Scaling Enable

Value	Name	Description
1	Enable	
0	Disable	The scaling factor is ignored and a scaling ratio of 1:1 is assumed.

6 Adaptive Filter for all Channels

Value	Name	Description
1	Enable Adaptive Filter on UV/RB Channels	8-tap Adaptive Filter Mode is on
0	Disable Adaptive Filter on UV/RB Channels	

5:4 AVS Filter Mode

Value	Name
0	5x5 Poly-phase filter + Bilinear (adaptive)
1	8x8 poly-phase filter + Bilinear (adaptive)
2	Bilinear filter only
3	Reserved

Programming Notes
In VD-to-SFC mode, value of 1 is not allowed.

3 Reserved

Format:	MBZ

2 IEF4Smooth_Enable

Value	Name	Description
0	**[Default]**	IEF is operating as a content adaptive detail filter based on 5x5 region.
1		IEF is operating as a content adaptive smooth filter based on 3x3 region

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.

1 Skin Tone Tuned IEF_Enable

Exists If:	//IEF Enable = 1

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.

SFC_STATE

0 — IEF Enable

Value	Name	Description
1	Enable	IEF Filter is Enabled
0	Disable	IEF Filter is Disabled

Restriction
Restriction : For Integral Image Mode and VD Mode, this field is Reserved and MBZ.

5 — 31:28 — Reserved

Format:	MBZ

5 — 27:16 — Source Region Height

Format:	U12-1

Source/Crop Region Height Minus 1 of the Input Frame in Unit of Pixel [11:0].
This field specifies the source/crop region of the input frame used for scaling of the graphic view. It defines the out-of-frame boundary used prior to AVS/IEF interpolation operation. The max value should be programmed to be equal or small than the input FrameHeightinMBminus 1 field. e.g. for 1920x1080 content, FrameHeightInMBsMinus1 is equal to1087 (1088 lines); however, the crop region height should be set to 1079(1080 lines). The last 8 lines are assumed to be not usable and should not be used as source pixels for Scaling or IEF operations. Otherwise, the bad pixels will breach and cause artifacts into the scaled output frame.

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.
Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422/444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.

5 — 15:12 — Reserved

Format:	MBZ

5 — 11:0 — Source Region Width

Format:	U12-1

Source/Crop Region Width Minus 1 of the Input Frame in Unit of Pixel [11:0].
This field specifies the source/crop region of the input frame used for scaling of the graphic view. It defines the out-of-frame boundary used prior to AVS/IEF interpolation operation. The max value should be programmed to be equal or small than the input FrameWidthinMBminus 1 field. e.g. for 1920x1080 content, FrameWidthInMBsMinus1 is equal to1919 (1920 pixel wide); however, the crop region width should be set to less than 1909(1910 pixel wide). The last 10 pixels of the frame are assumed to be not usable and should not be used as source pixels for Scaling or IEF operations. Otherwise, the bad pixels will breach and cause artifacts into the scaled output frame.

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.

SFC_STATE

		Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422 - multiple of 2. 444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.
6	31:28	**Reserved** Format: MBZ
	27:16	**Source Region Vertical Offset** Format: U12 **Vertical Offset Of The SRC Region Relative To The Starting Position Of The Input Frame In Unit Of Pixel [11:0]** This field specifies the vertical offset of the starting position of the scaled region relatives to the starting position (pixel 0,0) of the output frame. It defines the out-of-frame boundary used prior to AVS/IEF interpolation operation. This value should be set to zero if the starting corner of the crop region is same as the input frame region. The sum of this value and the src/crop region size heightminus1 must be programmed to be equal or small than the input FrameHeightinMBminus 1 field. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ. Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422/444/400 - no restrictions.
	15:12	**Reserved** Format: MBZ
	11:0	**Source Region Horizontal Offset** Format: U12 **Horizontal Offset Of The SRC Region Relative To The Starting Position Of The Input Frame In Unit Of Pixel [11:0]** This field specifies the horizontal offset of the starting position of the scaled region relatives to the starting position (pixel 0,0) of the output frame. It defines the out-of-frame boundary used prior to AVS/IEF interpolation operation. This value should be set to zero if the starting corner of the crop region is same as the input frame region. The sum of this value and the src/crop region size widthminus1 must be programmed to be equal or small than the input FrameWidthinMBminus 1 field. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ. Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422 - multiple of 2. 444/400 - no restrictions.

SFC_STATE

7	31:28	**Reserved** Format: MBZ
	27:16	**Output Frame Height** Format: U12-1 It is set to the value of the final output resolution of the graphic view. Since the max value support is 4k pixels, the max value allowed is 4K minus 1. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ. Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422/444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.
	15:12	**Reserved** Format: MBZ
	11:0	**Output Frame Width** Format: U12-1 It is set to the value of the final output resolution of the graphic view. Since the max value support is 4k pixels, the max value allowed is 4K minus 1. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ. Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422 - multiple of 2. 444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.
8	31:28	**Reserved** Format: MBZ
	27:16	**Scaled Region Size Height** Format: U12-1 Multiple of 2 Pixels It is set to the height of the scaled region over the output frame of the graphic view. **Programming Notes** The Max Value =< [The Output Frame Height Minus1]. **Restriction** Restriction : For AVS mode, if rotation_mode = 0/180, the restriction is tied to chroma output format type: 420 - multiple of 2. 422/444/400 - no restrictions.

SFC_STATE

		For AVS mode, if rotation_mode = 90/270, the restriction is tied to chroma output format type: 420/422 - multiple of 2. 444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.
	15:12	**Reserved** Format: MBZ
	11:0	**Scaled Region Size Width** Format: U12-1 Multiple of 2 Pixels It is set to the Width of the scaled region over the output frame of the graphic view. **Programming Notes** The Max Value =< [The Output Frame Width Minus1]. **Restriction** Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422 - multiple of 2. 444/400 - no restrictions. **Min Resolution** is 128 pixels. **Max Resolution** is 4K pixels.
9	31:29	**Reserved** Format: MBZ
	28:16	**Scaled Region Vertical Offset** Format: S12 **Vertical Offset (in pixels) Of The Scaled Region Relatives to The Starting Position Of The Output Frame In Unit Of Pixel [11:0]** This field specifies the vertical offset of the starting position of the scaled region relatives to the starting position (pixel 0,0) of the output frame. The gap between the scaled and output frame shall be filled by hardware with a set of programmed YUV/RGB values (Grey Bar). This value should be set to zero if the starting corner of the scaled region is same as the output frame region. The sum of this value and the scaled region size Heightminus1 must be programmed to be equal or small than the output FrameHeightinMBminus 1 field plus 16. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ. Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422/444/400 - no restrictions. Restriction : This field must be set to zero if SFC Ouput surface format type is NV12.
	15:13	**Reserved** Format: MBZ
	12:0	**Scaled Region Horizontal Offset** Format: S12

SFC_STATE

Horizontal Offset (in pixels) Of The Scaled Region Relatives to The Starting Position Of The Output Frame In Unit Of Pixel [11:0]

This field specifies the horizontal offset of the starting position of the scaled region relatives to the starting position (pixel 0,0) of the output frame. The gap between the scaled and output frame shall be filled by hardware with a set of programmed YUV/RGB values (Grey Bar). This value should be set to zero if the starting corner of the scaled region is same as the output frame region. The sum of this value and the scaled region size Widthminus1 must be programmed to be equal or small than the output FrameWidthinMBminus 1 field plus 16.

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.
Restriction : For AVS mode, the restriction is tied to chroma output format type: 420 - multiple of 2. 422 - multiple of 2. 444/400 - no restrictions.
Restriction : This field must be set to zero if SFC Ouput surface format type is NV12.

10	31:26	**Reserved** Format: MBZ
	25:16	**Gray Bar Pixel - Y/R** Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This is the default value used to fill in the area between the scaled region and the output frame size (aka Gray Bar) in Y or R channel on the AYUV or RGBA domain respectively. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
	15:10	**Reserved** Format: MBZ
	9:0	**Gray Bar Pixel - U/G** Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This is the default value used to fill in the area between the scaled region and the output frame size (aka Gray Bar) in U or G channel on the AYUV or RGBA domain respectively. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
11	31:26	**Reserved** Format: MBZ

SFC_STATE

	25:16	**Gray Bar Pixel - V/B** Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This is the default value used to fill in the area between the scaled region and the output frame size (aka Gray Bar) in V or B channel on the AYUV or RGBA domain respectively. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
	15:10	**Reserved** Format: MBZ
	9:0	**Gray Bar Pixel - A** Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This is the default value used to fill in the area between the scaled region and the output frame size (aka Gray Bar) in A channel on the AYUV or RGBA domain respectively. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
12	31:26	**Reserved** Format: MBZ
	25:16	**UV Default value for V channel (For Mono Input Support)** Exists If: //Input NOT originated by VEBOX. Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This field specifies the UV default value fill in to the UV output channels when input from VDBOX is set to Monochrome. **Restriction** Restriction : Not used when input is originated by VEBOX (Including Integral Image Mode).
	15:10	**Reserved** Format: MBZ
	9:0	**UV Default value for U channel (For Mono Input Support)** Exists If: //Input NOT originated by VEBOX. Format: 10-bit UNORM Type

SFC_STATE

		Range:[0.0, +1.0] This field specifies the UV default value fill in to the UV output channels when input from VDBOX is set to Monochrome. **Restriction** Restriction : Not used when input is originated by VEBOX (Including Integral Image Mode).
13	31:10	**Reserved** Format: MBZ
	9:0	**Alpha Default Value** Format: 10-bit UNORM Type **Range:**[0.0, +1.0] This field specifies the Alpha default value fill into the alpha output channel when output format type is set to RGBA8/10. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
14	31:21	**Reserved** Format: MBZ
	20:0	**Scaling Factor Height** Format: U4.17 This field specifies the scaling ratio of the vertical sizes between the crop/source region and the scaled region. The destination pixel coordinate, y-axis, is multiplied with this scaling factor to mapping back to the source input pixel coordinate. The field specifies the ratio of crop height resolution/ scaled height resolution. This implies $1/sf_v$ in the equation. **Value** / **Name** / **Description** 0 / / Reserved **Restriction** Restriction : Integral Image Mode supports downsampling only.
15	31:21	**Reserved** Format: MBZ
	20:0	**Scaling Factor Width** Format: U4.17 This field specifies the scaling ratio of the horizontal sizes between the crop/source region and the scaled region. The destination pixel coordinate, x-axis, is multiplied with this scaling factor to mapping back to the source input pixel coordinate. The field specifies the ratio of crop width resolution/ scaled width resolution. This implies $1/sf_h$ in

SFC_STATE

the equations above.

Value	Name	Description
0		Reserved

Restriction
Restriction : Integral Image Mode supports downsampling only.

16	31:0	**Reserved** Format: MBZ
17	31:12	**Output Frame Surface Base Address** Specifies the 4K byte aligned frame buffer address for outputting the scaled up/down image. Data is stored in Tile-Y format. For Integral Image mode, the accumulated integral image values will be packed linear in this surface. **Programming Notes** This field is ignored if I-frame only mode is set to 0 (Disable).
	11:0	**Reserved** Format: MBZ
18	31:16	**Reserved** Format: MBZ
	15:0	**Output Frame Surface Base Address High** This field is for the upper range [47:32] of Output Frame Surface Base Address. For Integral Image mode, the accumulated integral image values will be packed linear in this surface.
19	31:15	**Reserved** Format: MBZ
	14:13	**Output Surface Tiled Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

SFC_STATE

12 Output Frame Surface Base Address - Row Store Scratch Buffer Cache Select

Format:	MBZ

Value	Name	Description
0	Disable **[Default]**	This field must be programmed to 0

Programming Notes
This must be set to 0

11 Reserved

Format:	MBZ

10 Output Frame Surface Base Address - Memory Compression Mode

Format:	U1

Description
Distinguishes vertical from horizontal compression. Please refer to vol1a**Memory Data Formats chapter - section**media Memory Compression for more details.

Value	Name
0	Vertical Compression
1	Horizontal Compression

9 Output Frame Surface Base Address - Memory Compression Enable

Format:	Enable

Memory compression will be attempted for this surface.

8:7 Output Frame Surface Base Address - Arbitration Priority Control

Format:	HEVC_ARBITRATION_PRIORITY

6:1 Output Frame Surface Base Address - Index to Memory Object Control State (MOCS) Tables

Format:	U6

The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers.
The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.

0 Reserved

SFC_STATE

20	31:12	**AVS Line Buffer Surface Base Address** Specifies the 4K byte aligned frame buffer address for scratch space used for row/column store. This surface is used only if the internal buffer inside the SFC HW is not large enough to contain all row/column memory accesses. The AVS line buffer needs to be a valid address even for 1:1 scaling if SFC is used.
	11:0	**Reserved** Format: MBZ
21	31:16	**Reserved** Format: MBZ
	15:0	**AVS Line Buffer Surface Base Address High** This field is for the upper range [47:32] of AVS Line Buffer Surface Base Address. AVS Line buffer address needs to be valid even for 1:1 scaling if SFC is used.
22	31:15	**Reserved** Format: MBZ
	14:13	**AVS Line Buffer Tiled Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

	12	**AVS Line Buffer Base Address - Row Store Scratch Buffer Cache Select** Format: U1 This field controls if the Row Store is going to store inside Media Cache (rowstore cache) or to LLC.

Value	Name	Description
0	LLC **[Default]**	Buffer going to LLC

Programming Notes
This suface does not support to put in Row Store Scratch Buffer. Must be set to 0

	11	**Reserved** Format: MBZ

SFC_STATE

	10	**AVS Line Buffer Base Address - Memory Compression Mode** Default Value: 0 Horizontal Compression Mode Format: U1 Distinguishes vertical from horizontal compression. Please refer to vol1a **Memory Data Formats chapter - section** media Memory Compression for more details. **Programming Notes** Memory compression is not supported. This bit is not used. Default to 0
	9	**AVS Line Buffer Base Address - Memory Compression Enable** Default Value: 0 Disable Format: Enable This bit control memory compression for this surface **Programming Notes** This bit must be set to 0 (Memory compression is not supported in this surface)
	8:7	**AVS Line Buffer Base Address - Arbitration Priority Control** Format: HEVC_ARBITRATION_PRIORITY
	6:1	**AVS Line Buffer Base Address - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
23	31:12	**IEF Line Buffer Surface Base Address** Specifies the 4K byte aligned frame buffer address for the scratch space used for row/column store. This surface is used only if the internal buffer inside SFC HW is not large enough to contain all row/column memory accesses. **Restriction** Restriction : For Integral Image Mode, this field is Reserved and MBZ.
	11:0	**Reserved** Format: MBZ
24	31:16	**Reserved** Format: MBZ

SFC_STATE

15:0 IEF Line Buffer Surface Base Address High

This field is for the upper range [47:32] of IEF Line Buffer Surface Base Address.

Restriction
Restriction : For Integral Image Mode, this field is Reserved and MBZ.

25 31:15 Reserved

Format:	MBZ

14:13 IEF Line Buffer Tiled Mode

Format:	U2

For Media Surfaces:

This field specifies the tiled resource mode.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

12 IEF Line Buffer Base Address - Row Store Scratch Buffer Cache Select

Format:	U1

This field controls if the Row Store is going to store inside Media Cache (rowstore cache) or to LLC.

Value	Name	Description
0	LLC **[Default]**	Buffer going to LLC

Programming Notes
This surface does not support Rowstore Scratch Buffer Cache. Must be programmed to 0

11 Reserved

Format:	MBZ

10 IEF Line Buffer Base Address - Memory Compression Mode

Default Value:	0
Format:	U1

Distinguishes vertical from horizontal compression. Please refer to vol1a **Memory Data Formats chapter - section** media Memory Compression for more details.

Programming Notes
Must be zero; memory compression is not supported for this surface. Default to0

9 IEF Line Buffer Base Address - Memory Compression Enable

Default Value:	0 Disable
Format:	Enable

SFC_STATE

		Programming Notes Memory compression is not supported for this surface Must be 0.
	8:7	**IEF Line Buffer Base Address - Arbitration Priority Control** Format: HEVC_ARBITRATION_PRIORITY
	6:1	**IEF Line Buffer Base Address - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
26	31:0	**Reserved** Format: MBZ
27	31:0	**Reserved** Format: MBZ
28	31:0	**Reserved** Format: MBZ
29	31:28	**Output Surface Format**
	27	**Output Surface Interleave Chroma Enable**
	26:20	**Reserved** Format: MBZ
	19:3	**Output Surface Pitch** Format: U17-1 Pitch in (Bytes - 1) This field specifies the surface pitch.

Value	Name	Description
[0,2047]	SURFTYPE_BUFFER Surfaces	[1B, 2048B]
[0, 524287]	Other Linear Surfaces	[64B, 512KB] = [1 CL, 8K CLs]
[511, 524287]	X-tiled Surface	[512B, 256KB] = [1tile, 512 tiles]
[127, 524287]	Y-tiled surfaces	[128B,256KB] = [1 tile, 2048 tiles]

Programming Notes

- For tiled surfaces, the pitch must be a multiple of the tile width
- For Linear surfaces, the pitch must be a multiple of CL (64B) width
- If **Half Pitch for Chroma** is set, this field must be a multiple of two tile widths for tiled

SFC_STATE

surfaces, or a multiple of 2 bytes for linear surfaces.

If Media Memory Compression is enabled, the following max pitch size restriction must be honored. For larger resolution, Media Memory compression Must be disabled.

Tiling Mode	Pixel Format	Max Frame Width (bytes)	Max Frame Width (pixels)	Max Pitch (bytes)
Legacy 4K	8bpp	16k	16k	16k + 127
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 127
	128bpp	16k	1k	16k + 127
TileYF	8bpp	8k	8k	8k + 63
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 255
	128bpp	16k	1k	16k + 255
TileYS	8bpp	16k	16k	16k + 255
	16bpp	16k	8k	16k + 511
	32bpp	16k	4k	16k + 511
	64bpp	16k	2k	16k + 1023
	128bpp	16k	1k	16k + 1023

2

Output Surface Half Pitch For Chroma

Exists If:	//PLANAR Surface Formats Only
Format:	Enable

This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field.

1

Output Surface Tiled

Format:	Boolean

This field specifies whether the surface is tiled.

Value	Name	Description
1	True	Tiled
0	FALSE	Linear

Programming Notes

- Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory.

SFC_STATE

- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
- The tiled surfaces of current picture and reference picture should be declared as the identical type in VDI mode with the identical Height, Width and Format.

0 — Output Surface Tile Walk

Format:	SFC_Tile_Walk

This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions.

Value	Name
0	TILEWALK_XMAJOR
1	TILEWALK_YMAJOR

Programming Notes

- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.

This field is ignored when the surface is linear.

31:30 — Reserved

Format:	MBZ

29:16 — Output Surface X Offset For U

Exists If:	//PLANAR Surface Formats Only
Format:	U14 Pixel Offset

This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled.

Programming Notes

For PLANAR_420 and PLANAR_422 surface formats, this field must be zero.

15:14 — Reserved

Format:	MBZ

13:0 — Output Surface Y Offset For U

Exists If:	//PLANAR Surface Formats Only
Format:	U14 Pixel Row Offset

This field specifies the veritical offset in rows from the **Surface Base Address** to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled.

Programming Notes

For PLANAR_420 and PLANAR_422 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs.

31:30 — Reserved

Format:	MBZ

<table>
<tr><th colspan="4">SFC_STATE</th></tr>
<tr><td></td><td>29:16</td><td colspan="2">Output Surface X Offset For V<table><tr><td>Exists If:</td><td>//PLANAR Surface Formats with Interleaved Chroma Disable</td></tr><tr><td>Format:</td><td>U14 Pixel Offset</td></tr></table>This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the V(Cr) plane.<table><tr><th>Programming Notes</th></tr><tr><td>For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td colspan="2">Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>13:0</td><td colspan="2">Output Surface Y Offset For V<table><tr><td>Exists If:</td><td>//PLANAR Surface Formats with Interleaved Chroma Disable</td></tr><tr><td>Format:</td><td>U14 Pixel Offset</td></tr></table>This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane.<table><tr><th>Programming Notes</th></tr><tr><td>For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.</td></tr></table></td></tr>
<tr><td>32</td><td>31:1</td><td colspan="2">Reserved</td></tr>
<tr><td>33</td><td>31:0</td><td colspan="2">Reserved</td></tr>
</table>

Shift Left

shl - Shift Left

Source: Eulsa

Length Bias: 4

Description

Perform component-wise logical left shift of the bits in src0 by the shift count indicated in src1, storing the results in dst, inserting zero bits in the number of LSBs indicated by the shift count.
Hardware detects overflow properly and uses it to perform any saturation operation on the result, as long as the shifted result is within 33 bits. Otherwise, the result is undefined.
Note: For word and DWord operands, the accumulators have 33 bits.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.

Format:
[(pred)] shl[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Accumulator cannot be destination, implicit or explicit.

Syntax

```
[(pred)] shl[.cmod] (exec_size) reg reg reg [(pred)] shl[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F dst.chan[n] = src0.chan[n] « shiftCnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*W,*D	*W,*D
*W,*D	*Q
*Q	*W,*D
*Q	*Q

DWord	Bit	Description
0..3	127:64	**RegSource**

shl - Shift Left

		Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_REG**
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_IMM**
	63:32	**Operand Controls** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Shift Right

shr - Shift Right

Source: Eulsa

Length Bias: 4

Description

Perform component-wise logical right shift with zero insertion of the bits in src0 by the shift count indicated in src1, storing the results in dst. Insert zero bits in the number of MSBs indicated by the shift count.
src0 and dst can have different types and can be signed or unsigned.
Note: For word and DWord operands, the accumulators have 33 bits.
Note: For unsigned src0 types, shr and asr produce the same result.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.

Format:
[(pred)] shr[.cmod] (exec_size) dst src0 src1

Syntax

```
[(pred)] shr[.cmod] (exec_size) reg reg reg [(pred)] shr[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F dst.chan[n] = src0.chan[n] » shiftCnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
UB, UW, UD	UB, UW, UD
UW, UD	UW, UD
UW, UD	UQ
UQ	UW, UD
UQ	UQ

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

shr - Shift Right

	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

SIMD8 Render Target Read MSD

MSD_RTR_SIMD8 - SIMD8 Render Target Read MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message **Programming Notes** This field must be programmed to 0
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.

MSD_RTR_SIMD8 - SIMD8 Render Target Read MSD

	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 0Dh Format: Opcode Render Target Read message
	13	**Per-Sample PS Enable** Format: Enable If set, PS reads color phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Reserved** Format: MBZ Ignored
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:9	**Reserved** Format: MBZ Ignored

MSD_RTR_SIMD8 - SIMD8 Render Target Read MSD

	8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD8 single source message. Use slots [7:0] for the pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

SIMD8 Render Target Write MSD

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

17:14 **Message Type**

Default Value:	0Ch
Format:	Opcode

Render Target Write message

13 **Per-Sample PS Enable**

Format:	Enable

If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.

Programming Notes

This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.

When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.

When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).

When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.

12 **Last Render Target Select**

Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set.

In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 **Slot Group Select**

Format:	MDC_RT_SGS

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 4h Format: Opcode SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

SIMD16 Render Target Read MSD

MSD_RTR_SIMD16 - SIMD16 Render Target Read MSD

Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message **Programming Notes** This field must be programmed to 0
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.

MSD_RTR_SIMD16 - SIMD16 Render Target Read MSD

	18	**Reserved** Format: MBZ Ignored
	17:14	**Message Type** Default Value: 0Dh Format: Opcode Render Target Read message
	13	**Per-Sample PS Enable** Format: Enable If set, PS reads color phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Reserved** Format: MBZ Ignored
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:9	**Reserved** Format: MBZ Ignored

MSD_RTR_SIMD16 - SIMD16 Render Target Read MSD

	8	**Render Target Message Subtype** Default Value: 0h Format: Opcode SIMD16 single source message. Use slots [15:0] for the pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

SIMD16 Render Target Write MSD

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Format: MBZ Ignored
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Format: MBZ Ignored

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

17:14 **Message Type**

Default Value:	0Ch
Format:	Opcode

Render Target Write message

13 **Per-Sample PS Enable**

Format:	Enable

If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.

Programming Notes

This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.

When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.

When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).

When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.

12 **Last Render Target Select**

Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set.

In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 **Slot Group Select**

Format:	MDC_RT_SGS

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 0h Format: Opcode SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Split Send Message

sends - Split Send Message

Source: Eulsa

Length Bias: 4

Description

The sends instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The sends instruction adds an entry to the EU's message request queue. The request message is stored in a block of contiguous GRF registers. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src0> and <src1> are the lead GRF registers for the first and second block of the request respectively. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers corresponding to src0) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend message descriptor field <ex_desc> contains the target function ID, the Extended Message Length (the number of consecutive GRF registers corresponding to src1) and the extended function control signals. WrEn is forwarded to the target function in the message sideband.

The sends instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.

Message descriptor field <desc> can be a 32-bit immediate, imm32, or a 32-bit scalar register, <reg32a>. GEN restricts that the 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0<0;1,0>:ud. When <desc> is a register operand, only the lower 31 bits of <reg32a> are used.

Extended Message descriptor field <ex_desc> can be a 32-bit immediate, imm32 or a 32bit scalar register, <reg32a>. The bits3:0 of the <ex_desc> specifies the SFID for the message.The EOT field always comes from bit127 of the instruction word, which is the bit5 of <ex_desc>. A thread must terminate with a sends instruction with EOT turned on. The bits9:6 of <ex_desc> specify the extended message length and bits31:12 specify the 20bit extended function control. Interpretation of the extended function control signals is subject to the target external function. The scalar register <reg32a> is selected when SelReg32ExDesc is set, ExDesc.RegNum[3:0] provides the addressing for reg32a for extended message descriptor. This selects one of the index sub registers. Subregisters selected are always aligned to dword. This implies, the even index subregisters must be used.</reg32a></ex_desc></ex_desc></ex_desc></reg32a></ex_desc>

Function control is now extended to 20 bits as specified in the below definition.

<src0> is a 256-bit aligned GRF register. It serves as the leading GRF register of the request.
<src1> is a 256-bit aligned GRF register or a null register. It serves as the leading GRF register for the second block of the request when it is not a null register. It is required that the second block of GRFs does not overlap with the first block. If it is a null register the Extended Message Length must be 0. The sum of Message Length and Extended Message Length must not be greater than 15.
The source dependency control, {NoSrcDepSet} is used to control the setting of source dependency for both the sources.
<dest> serves for two purposes: to provide the leading GRF register location for the response message if

sends - Split Send Message

present, and to provide parameters to form the channel enable sideband signals.
<dest> signals whether there is a response to the message request. It can be either a null register, a direct-addressed GRF register or a register-indirect GRF register. Otherwise, hardware behavior is undefined.
If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null.
If <dest> is a GRF register, the register number is forwarded to the shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off.
The destination type field is always valid and is used to generate the WrEn. This is true even if <dest> is a null register (this is an exception for null as for most cases these fields are ignored by hardware).
The address immediates for indirect sources and destination must be oword aligned.
The 16-bit channel enables of the message sideband are formed based on the WrEn. Interpretation of the channel enable sideband signals is subject to the target external function. In general for a 'sends' instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases.
NoDDClr and NoDDChk must not be used for send instruction.

Send a message stored in GRF locations starting at <src0> followed by <src1> to a shared function identified by <ex_desc> along with control from <desc> and <ex_desc> with a GRF writeback location at <dest>.

Format:
[(pred)] sends (exec_size) <dest> <src0> <src1> <ex_desc> <desc>

Restriction

Restriction : Software must obey the following rules in signaling the end of thread using the sends instruction: The posted destination operand must be null. No acknowledgement is allowed for the sends instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource. A thread must terminate with a sends instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a sends instruction with message to the following shared functions: Sampler unit, NULL function For example, a thread may terminate with a URB write message or a render cache write message. A root thread originated from the media (generic) pipeline must terminate with a sends instruction with message to the Thread Spawner unit. A child thread should also terminate with a sends to TS. Please refer to the Media Chapter for more detailed description. The sends instruction can not update accumulator registers. Saturate is not supported for sends instruction. ThreadCtrl encodings Switch is not supported for sends instruction. The sends with EOT should use register space R112-R127 for <src>. This is to enable loading of a new thread into the same slot while the message with EOT for current thread is pending dispatch. Any instruction updating the ARF must use a {Switch} if the ARF is not used before EOT.</src>

Syntax

```
[(pred)] sends (exec size) reg greg greg imm32 imm32 [(pred)] sends (exec size)
reg greg greg imm32 reg32a [(pred)] sends (exec_size) reg greg reg reg32a imm32
[(pred)] sends (exec_size) reg greg reg reg32a reg32a
```

sends - Split Send Message

Pseudocode

```
Evaluate(WrEn); <MsgChEnable> = WrEn; <SourceReg0> = <src0>.RegNum; <SourceReg1>
= <src1>.RegNum; MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg0>,
<SourceReg1>,<ex_desc>, <dest>);
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95:80	**ExDesc[31:16]** Format: ExtMsgDescpt[31:16]
	79	**Source 0 Addressing Mode** Format: AddrMode
	78	**Reserved** Exists If: ([Source 0 Addressing Mode]=='Direct') Format: MBZ
	78	**Source 0 Address Immediate Sign [9]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[9]
	77	**SelReg32Desc**
	76:73	**Source 0 Address Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Indirect')
	76:69	**Source 0 Register Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	72:68	**Source 0 Address Immediate [8:4]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[8:4]
	68	**Source 0 Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	67:64	**ExDesc[9:6]** Format: ExtMsgDescpt[9:6]
	63	**Destination Addressing Mode** Format: AddrMode

sends - Split Send Message

	62	**Destination Address Immediate Sign [9]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[9]
	62	**Reserved** Exists If: ([Destination Addressing Mode]=='Direct') Format: MBZ
	61	**Reserved** Format: MBZ
	60:57	**Destination Address Subregister Number** Exists If: ([Destination Addressing Mode]=='Indirect')
	60:53	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct')
	56:52	**Destination Address Immediate [8:4]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[8:4]
	52	**Destination Subregister Number [4]** Exists If: ([Destination Addressing Mode]=='Direct')
	51:44	**Source 1 Register Number**
	43:41	**Reserved** Format: MBZ
	40:37	**Destination Type**
	36	**Source 1 Register File** Format: RegFile[0]
	35	**Destination Register File** Format: RegFile[0]
	34	**MaskCtrl**
	33:32	**Flag Register Number/Subregister Number**
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Shared Function ID (SFID)** Format: SFID
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ

sends - Split Send Message		
	6:0	**Opcode** Format: EU_OPCODE

STATE_BASE_ADDRESS

STATE_BASE_ADDRESS

Source: BSpec

Length Bias: 2

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE.
For more information see the Base Address Utilization table in the Memory Access Indirection narrative topic.

Programming Notes

The following commands must be reissued following any change to the base addresses:

- 3DSTATE_CC_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- 3DSTATE_SAMPLER_STATE_POINTERS
- 3DSTATE_VIEWPORT_STATE_POINTERS
- MEDIA_STATE_POINTERS

Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set before programming STATE_BASE_ADDRESS command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies in certain workload programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be traded off against performance by not implementing this programming note.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 01h STATE_BASE_ADDRESS
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description

STATE_BASE_ADDRESS

		11h	DWORD_COUNT_n **[Default]**	Excludes DWord (0,1)
		10h	DWORD_COUNT_modify	Excludes DWord (0,1) To avoid updating the "Bindless Surface State Base Size" when the "Bindless Surface State Base Address Modify Enable" bit is cleared then the size of the command must be reduced to one less than the size of the instruction.

1..2 — 63:12 — General State Base Address

Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for general state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

Programming Notes

Bounds checking is performed on general state accesses by Data Port Shared Functions for stateless A32 messages.
Bounds checking is enabled when General State Base Address [46:12] + General State Buffer Size [31:12] is <= 2^47. This ensures that the General State Buffer does not straddle the canonical address boundary where GraphicsAddress [47] changes.

Restriction

Restriction :
General State Base Address [47:12] + General State Buffer Size [31:12] must be < 2^48. It is illegal programming for this to be >= 2^48.

Restriction : When using stateless (A32) Data Port messages, General State Base Address [47:12] + Buffer Base Address [31:0] must be < 2^48. It is illegal for this to be >= 2^48.

11 — Reserved

Format:	MBZ

10:4 — General State Memory Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect state using the **General State Base Address**, with the exception of the stateless data port accesses.

3:1 — Reserved

Format:	MBZ

0 — General State Base Address Modify Enable

Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated address.
1h	Enable	Modify the address.

STATE_BASE_ADDRESS

3	31:23	**Reserved** Format: MBZ
	22:16	**Stateless Data Port Access Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for stateless data port accesses.
	15:0	**Reserved** Format: MBZ
4..5	63:12	**Surface State Base Address** Format: GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for binding table and surface state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Format: MBZ
	10:4	**Surface State Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Surface State Base Address**.
	3:1	**Reserved** Format: MBZ
	0	**Surface State Base Address Modify Enable** Format: Enable The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated address.
1h	Enable	Modify the address.

Programming Notes
Setting this bit to 1 in a batch buffer causes the resource streamer to stop; for performance reasons the SW should only place commands with this bit set in the ring buffer.
Before programming the Surface State Base Address, the RS must be disabled. Within a batch buffer where the RS is enabled, RS may be disabled thru a MI_RS_CONTROL command with Resource Streamer Control cleared prior to the STATE_BASE_ADDRESS with Surface State Base Address Modify Enable set and then re-enabled with another MI_RS_CONTROL with Resource Streamer Control set.

STATE_BASE_ADDRESS

6..7 | 63:12 | Dynamic State Base Address

Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for sampler and viewport state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

11 | Reserved

Format:	MBZ

10:4 | Dynamic State Memory Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect state using the **Dynamic State Base Address**. Push constants defined in 3DSTATE_CONSTANT_(VS | GS | PS) commands do not use this control state, although they can use the corresponding base address. The memory object control state for push constants is defined within the command.

3:1 | Reserved

Format:	MBZ

0 | Dynamic State Base Address Modify Enable

Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated address.
1h	Enable	Modify the address.

8..9 | 63:12 | Indirect Object Base Address

Format:	GraphicsAddress[63:12]IndirectObject

Specifies the 4K-byte aligned base address for indirect object load in MEDIA_OBJECT command.

11 | Reserved

Format:	MBZ

10:4 | Indirect Object Memory Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect objects using the **Indirect Object Base Address**.

3:1 | Reserved

Format:	MBZ

0 | Indirect Object Base Address Modify Enable

Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

STATE_BASE_ADDRESS

		Value / **Name** / **Description** 0h / Disable / Ignore the updated address. 1h / Enable / Modify the address.
10..11	63:12	**Instruction Base Address** Format: GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for all EU instruction accesses. GraphicsAddress[63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Format: MBZ
	10:4	**Instruction Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for EU instructions using the **Instruction Base Address**.
	3:1	**Reserved** Format: MBZ
	0	**Instruction Base Address Modify Enable** Format: Enable The other fields in this DWord and the following DWord are updated only when this bit is set. **Value** / **Name** / **Description** 0h / Disable / Ignore the updated address. 1h / Enable / Modify the address.
12	31:12	**General State Buffer Size** Format: U20 FormatDesc This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.
	11:1	**Reserved** Format: MBZ
	0	**General State Buffer Size Modify Enable** Format: Enable The bound in this DWord is updated only when this bit is set. **Value** / **Name** / **Description** 0h / Disable / Ignore the updated bound.

STATE_BASE_ADDRESS

		1h	Enable	Modify the updated bound.

13 — 31:12 — Dynamic State Buffer Size

Format:	U20

FormatDesc
This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.

13 — 11:1 — Reserved

Format:	MBZ

13 — 0 — Dynamic State Buffer Size Modify Enable

Format:	Enable

FormatDesc
The bound in this DWord is updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated bound.
1h	Enable	Modify the updated bound.

14 — 31:12 — Indirect Object Buffer Size

Format:	U20

FormatDesc
This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.

14 — 11:1 — Reserved

Format:	MBZ

14 — 0 — Indirect Object Buffer Size Modify Enable

Format:	Enable

FormatDesc
The bound in this DWord is updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated bound.
1h	Enable	Modify the updated bound.

STATE_BASE_ADDRESS

15 — 31:12 — Instruction Buffer Size

Format:	U20

FormatDesc
This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.

11:1 — Reserved

Format:	MBZ

0 — Instruction Buffer size Modify Enable

Format:	Enable

FormatDesc
The bound in this DWord is updated only when this bit is set.

Value	Name	Description
0h	Disable	Ignore the updated bound.

16..17 — 63:12 — Bindless Surface State Base Address

Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for bindless surface state accesses.
GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

11 — Reserved

Format:	MBZ

10:4 — Bindless Surface State Memory Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect state using the **Bindless Surface State Base Address**.

3:1 — Reserved

Format:	MBZ

0 — Bindless Surface State Base Address Modify Enable

Format:	Enable

Description
The other fields in this DWord are updated only when this bit is set.

Value	Name	Description

STATE_BASE_ADDRESS

		0h	Disable	Ignore the updated address
		1h	Enable	Modify the address
18	31:12	**Bindless Surface State Size**		
		Format:		U20
		This field indicates the size-1 of the Bindless Surface State buffer in 64-Byte increments. Any SSO beyond this maximum size points to offset 0. Example: If the buffer contains 512 surface states, then this field must be programmed to 0x1FF (511 decimal).		
	11:0	**Reserved**		
		Format:		MBZ

STATE_PREFETCH

STATE_PREFETCH

Source: BSpec

Length Bias: 2

(This command is provided strictly for performance optimization opportunities, and likely requires some experimentation to evaluate the overall impact of additional prefetching.)

The STATE_PREFETCH command causes the GPE to attempt to prefetch a sequence of 64-byte cache lines into the GPE-internal cache ("L2 ISC") used to access EU kernel instructions and fixed/shared function indirect state data. While state descriptors, surface state, and sampler state are automatically prefetched by the GPE, this command may be used to prefetch data not automatically prefetched, such as: 3D viewport state; Media pipeline Interface Descriptors; EU kernel instructions.

Restriction
Due to know HW issue this command doesn't achieve its intended purpose and must not be exercised/programmed by SW.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 03h STATE_PREFETCH
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 0h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:6	**Prefetch Pointer** Format: GraphicsAddress[31:6] Specifies the 64-byte aligned address to start the prefetch from. This pointer is an absolute virtual address, it is not relative to any base pointer.
	5:3	**Reserved** Format: MBZ

STATE_PREFETCH

	2:0	**Prefetch Count** Format: U3-1 count of cache lines Indicates the number of contiguous 64-byte cache lines that will be prefetched.

Value	Name	Description
[0,7]		indicating a count of [1,8]

STATE_SIP

STATE_SIP

Source: BSpec

Length Bias: 2

The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 02h STATE_SIP
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 1h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1..2	63:4	**System Instruction Pointer** Format: InstructionBaseOffset[63:4]Kernel Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the Instruction Base Address. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** This portion of the command is not context save/restored. The context image may restore this command as a 2 dword command rather than a 3 dword command.
	3:0	**Reserved** Format: MBZ

Sum of Absolute Difference 2

sad2 - Sum of Absolute Difference 2

Source: Eulsa

Length Bias: 4

The sad2 instruction takes source data channels from src0 and src1 in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between src0 and src1 and stores the scalar result in the first channel of the 2-tuple in dst.

The results are also stored in the accumulator register. The destination operand and the accumulator maintain 16 bits per channel precision.

The destination register must be aligned to even word (DWord). The even words in the destination region will contain the correct data. The odd words are also written but with undefined values.

Format:

[(pred)] sad2[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Source operands cannot be accumulators.

Restriction : The execution size cannot be 1 as the computation requires at least two data channels.

Syntax

```
[(pred)] sad2[.cmod] (exec_size) reg reg reg [(pred)] sad2[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
     for ( n = 0; n < exec_size; n += 2 ) {
         if ( WrEn.chan[n] ) {
            dst.chan[n] = abs(src0.chan[n] - src1.chan[n])
                        + abs(src0.chan[n+1] - src1.chan[n+1]);
         }
     }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
B, UB	W, UW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

sad2 - Sum of Absolute Difference 2		
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_IMM**
	63:32	**Operand Controls** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

Sum of Absolute Difference Accumulate 2

sada2 - Sum of Absolute Difference Accumulate 2

Source: Eulsa

Length Bias: 4

The sada2 instruction takes source data channels from src0 and src1 in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between src0 and src1, adds the intermediate result with the accumulator value corresponding to the first channel, and stores the scalar result in the first channel of the 2-tuple in dst.

The destination operand and the accumulator maintain 16 bits per channel precision. Higher precision (guide bits) stored in the accumulator allows up to 64 rounds of sada2 instructions to be issued back to back without overflowing the accumulator.

The destination register must be aligned to even word (DWord). The even words in the destination region will contain the correct data. The odd words are also written but with undefined values.

Format:

[(pred)] sada2[.cmod] (exec_size) dst src0 src1

Restriction

Restriction : Source operands cannot be accumulators.

Restriction : The execution size cannot be 1 as the computation requires at least two data channels.

Syntax

```
[(pred)] sada2[.cmod] (exec_size) reg reg reg [(pred)] sada2[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 2 ) { uwTmp = abs(src0.chan[n] - src1.chan[n]) + abs(src0.chan[n+1] - src1.chan[n+1]); if ( WrEn.chan[n] ) { dst.chan[n] = uwTmp + acc[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
B, UB	W, UW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG

sada2 - Sum of Absolute Difference Accumulate 2

	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

Typed Surface Read MSD

MSD1R_TS - Typed Surface Read MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Surface R/W
Group:	Scattered Typed Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 05h Format: Opcode Typed Surface Read message
	13:12	**Slot Group** Format: MDC_SG3 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Typed Surface Write MSD

MSD1W_TS - Typed Surface Write MSD

Source:	DataPort 1
Length Bias:	1
Family:	Typed Surface R/W
Group:	Scattered Typed Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 0Dh Format: Opcode Typed Surface Write message
	13:12	**Slot Group** Format: MDC_SG3 Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Untyped Surface Read MSD

MSD1R_US - Untyped Surface Read MSD		
Source:	DataPort 1	
Length Bias:	1	
Family:	Untyped Surface R/W	
Group:	Scattered Untyped Surface R/W	
DWord	**Bit**	**Description**
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 01h Format: Opcode Untyped Surface Read message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Untyped Surface Write MSD

MSD1W_US - Untyped Surface Write MSD

Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 09h Format: Opcode Untyped Surface Write message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

URB Hword Dual Block Read MSD

MSD_UR_HWDB - URB Hword Dual Block Read MSD

Source: Read-Only DataPort

Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved** Format: MBZ Ignored
	15	**Swizzle Control** Format: Opcode Value / Name / Description: 1 / URB_INTERLEAVED **[Default]** / Use two URB entries (**URB Handle 0** and **URB Handle 1**).
	14:4	**Global Offset** Format: U11 Specifes the offset, in units of Hword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,1023]
	3:0	**URB Opcode** Format: Opcode Value / Name / Description: 2 / URB_READ_HWORD **[Default]** / URB Hword Read message

URB Hword Dual Block Write MSD

MSD_UW_HWDB - URB Hword Dual Block Write MSD

Source: Read-Only DataPort

Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved** Format: MBZ Ignored
	15	**Swizzle Control** Format: Opcode Value / Name / Description 1 / URB_INTERLEAVED **[Default]** / Use two URB entries (**URB Handle 0** and **URB Handle 1**).
	14:4	**Global Offset** Format: U11 Specifes the offset, in units of Hword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,1023]
	3:0	**URB Opcode** Format: Opcode Value / Name / Description 0 / URB_WRITE_HWORD **[Default]** / URB Hword Read message

URB Oword Block Write MSD

MSD_UW_OWB - URB Oword Block Write MSD

Source: Read-Only DataPort

Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved** Format: MBZ Ignored
	15	**Swizzle Control** Format: Opcode Value / Name / Description 0 / URB_NOSWIZZLE **[Default]** / Use a single URB entry (**URB Handle 0**).
	14:4	**Global Offset** Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Format: Opcode Value / Name / Description 1 / URB_WRITE_OWORD **[Default]** / URB Oword Write message

URB Oword Dual Block Read MSD

MSD_UR_OWDB - URB Oword Dual Block Read MSD

Source: Read-Only DataPort

Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved** Format: MBZ Ignored
	15	**Swizzle Control** Format: Opcode Value / Name / Description 1 / URB_INTERLEAVED **[Default]** / Use two URB entries (**URB Handle 0** and **URB Handle 1**).
	14:4	**Global Offset** Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Format: Opcode Value / Name / Description 3 / URB_READ_OWORD **[Default]** / URB Oword Read message

URB Oword Dual Block Write MSD

MSD_UW_OWDB - URB Oword Dual Block Write MSD

Source: Read-Only DataPort

Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Format: MBZ Ignored
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved** Format: MBZ Ignored
	15	**Swizzle Control** Format: Opcode Value / Name / Description 1 / URB_INTERLEAVED **[Default]** / Use two URB entries (**URB Handle 0** and **URB Handle 1**).
	14:4	**Global Offset** Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Format: Opcode Value / Name / Description 1 / URB_WRITE_OWORD **[Default]** / URB Oword Write message

VD_PIPELINE_FLUSH

VD_PIPELINE_FLUSH		
Source: VideoCS Length Bias: 2		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: Fh Extended command Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Format: OpCode
	20:16	**SubOpcodeB** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWORD_COUNT_n** Default Value: 0h Excludes DWord (0) Format: =n Total Length - 2
1	31:20	**Reserved** Format: MBZ
	19	**MFX pipeline command flush** Format: U1
	18	**Reserved**
	17	**VD-ENC pipeline command flush** Format: U1

VD_PIPELINE_FLUSH		
	16	**HEVC pipeline command flush** Format: U1
	15:5	**Reserved** Format: MBZ
	4	**VD command/message parser Done** Format: U1
	3	**MFX pipeline Done** Format: U1
	2	**Reserved**
	1	**VD-ENC pipeline Done** Format: U1
	0	**HEVC pipeline Done** Format: U1

VEB_DI_IECP

VEB_DI_IECP

Source: VideoEnhancementCS

Length Bias: 2

The VEB_DI_IECP command causes the VEBOX to start processing the frames specified by VEB_SURFACE_STATE using the parameters specified by VEB_DI_STATE and VEB_IECP_STATE. The processing can start and end at any 64 pixel column in the frame. If **Starting X** and **Ending X** are used to split the frame into sections, it should not be split into more than 4 sections.

Each VEB_DI_IECP command should be preceded by a VEB_STATE command and the input/output VEB_SURFACE_STATE commands.

Programming Notes

When DI is enabled, only the Current Frame skin scores are outputted to the Skin Score Output surface.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Opcode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpA** Default Value: 0h VEB_DI_IECP Format: OpCode
	20:16	**SubOpB** Default Value: 3h VEB_DI_IECP Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Excludes DWords 0, 1 Value: 14h — Name: **[Default]**

VEB_DI_IECP

1	31:30	**Reserved** Format: MBZ
	29:16	**Starting X** Offset from the beginning of the frame to start processing. Must be a multiple of 64 to guarantee that it starts on a column boundary.
	15:14	**Reserved** Format: MBZ
	13:0	**Ending X** Offset from the beginning of the frame to stop processing. Must be a multiple of 64 or equal to the Surface Width to guarantee that it ends on a column boundary. **Programming Notes** **Restriction:**Ending_X - Starting_X must be > = 64.
2	31:12	**Current Frame Input Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for reading current frame as input to either the DN/DI or IECP stage.
	11	**Reserved** Format: MBZ
	10:0	**Current Frame Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
3	31:30	**Current Frame Input Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 0 – Highest priority 1 – Second highest priority 2 – Third highest priority 3 – Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Current Frame Input Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address

VEB_DI_IECP

4	31:12	**Previous Frame Input Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for reading the denoised previous frame as input to the DN/DI stage. This field is ignored if **DN Enable**, **DI Enable**, and **Hot Pixel Filtering Enable** are set to 0 (disable).
	11	**Reserved** Format: MBZ
	10:0	**Previous Frame Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
5	31:30	**Previous Frame Input Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 0 – Highest priority 1 – Second highest priority 2 – Third highest priority 3 – Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Previous Frame Input Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
6	31:12	**STMM Input Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for reading the STMM / denoise history. This field is ignored if **DN Enable**, **DI Enable**, and **Hot Pixel Filtering Enable** are set to 0 (disable). **Programming Notes** This field is ignored if **Disable Temporal Denoise Filter** is set to 1.
	11	**Reserved** Format: MBZ
	10:0	**STMM Input Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS

VEB_DI_IECP

7	31:30	**STMM Input Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 0 – Highest priority 1 – Second highest priority 2 – Third highest priority 3 – Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**STMM Input Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
8	31:12	**STMM Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for writing the STMM / denoise history. This field is ignored if **DN Enable**, **DI Enable**, and **Hot Pixel Filtering Enable** are set to 0 (disable). **Programming Notes** This field is ignored if **Disable Temporal Denoise Filter** is set to 1.
	11	**Reserved** Format: MBZ
	10:0	**STMM Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
9	31:30	**STMM Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 0 – Highest priority 1 – Second highest priority 2 – Third highest priority 3 – Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**STMM Output Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address

VEB_DI_IECP

10	31:12	**Denoised Current Frame Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for writing the current frame after the DN stage. This field is ignored if both **DN Enable** and **Hot Pixel Filtering Enable** are set to 0 (disable).
	11	**Reserved** Format: MBZ
	10:0	**Denoised Current Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
11	31:30	**Denoised Current Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 0 — Name: Highest priority Value: 1 — Name: Second highest priority Value: 2 — Name: Third highest priority Value: 3 — Name: Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Denoised Current Frame Output Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
12	31:12	**Current Frame Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for writing the current frame output. The output is from DN/DI if IECP is disabled, or from IECP if enabled.
	11	**Reserved** Format: MBZ
	10:0	**Current Frame Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
13	31:30	**Current Frame Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value — Name

VEB_DI_IECP

		0 Highest priority 1 Second highest priority 2 Third highest priority 3 Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Current Frame Output Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
14	31:12	**Previous Frame Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for writing the previous frame output. This field is ignored if **DI Enable** is set to 0 (disable).
	11	**Reserved** Format: MBZ
	10:0	**Previous Frame Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
15	31:30	**Previous Frame Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 0 Highest priority 1 Second highest priority 2 Third highest priority 3 Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Previous Frame Output Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
16	31:12	**Statistics Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for writing block level FMD and DN statistics as well as the frame level ACE histogram and FMD frame level statistics.

VEB_DI_IECP

	11	**Reserved** Format: MBZ
	10:0	**Statistics Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
17	31:30	**Statistics Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 0 Highest priority; 1 Second highest priority; 2 Third highest priority; 3 Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Statistics Output Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address
18	31:12	**Alpha/Vignette Correction Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned frame buffer address for reading the Alpha surface or vignette correction.
	11	**Reserved** Format: MBZ
	10:0	**Alpha/Vignette Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
19	31:30	**Alpha/Vignette Correction Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 0 Highest priority; 1 Second highest priority; 2 Third highest priority; 3 Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Alpha/Vignette Correction Address High** Format: GraphicsAddress[47:32] Specifies bits 47:32 of the address

VEB_DI_IECP

20	31:12	**LACE/ACE/RGB Histogram Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned address for writing the LACE histograms, or ACE histogram, and/or RGB histogram.
	11	**Reserved** Format: MBZ
	10:0	**LACE/ACE/RGB Histogram Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS Please refer to the Surface Control Bits Table below.
21	31:30	**LACE/ACE/RGB Histogram Surface Arbitration Priority Control** Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
0	Highest priority
1	Second highest priority
2	Third highest priority
3	Lowest priority

	29:16	**Reserved** Format: MBZ
	15:0	**LACE/ACE/RGB Histogram Output Address High** Format: GraphicsAddress[47:32] Bits 47:32 of address.
22	31:12	**Skin Score Output Address** Format: GraphicsAddress[31:12] Specifies bits 31:12 of the 4K byte aligned address for writing the Skin score output in case enabled.
	11	**Reserved** Format: MBZ
	10:0	**Skin Score Output Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS Please refer to the Surface Control Bits Table below.

VEB_DI_IECP

23	31:30	**Skin Score Output Surface Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 0 – Highest priority 1 – Second highest priority 2 – Third highest priority 3 – Lowest priority
	29:16	**Reserved** Format: MBZ
	15:0	**Skin Score Output Address High** Format: GraphicsAddress[47:32] Bits 47:32 of address

VEBOX_STATE

VEBOX_STATE

Source: VideoEnhancementCS

Length Bias: 2

Description

This command controls the internal functions of the VEBOX. This command has a set of indirect state buffers:

- DN/DI state
- IECP general state
- IECP Gamut Expansion/Compression state
- IECP Gamut Vertex Table state
- Capture Pipe state

: Adds the LACE LUT Table as an indirect state buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ

VEBOX_STATE

	11:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description Eh / / (Excludes DWords 0, 1)
1	31:25	**State Surface Control Bits** All Indirect state buffers use state surface control bits, only exception being 3D LUT state buffer for which the state surface control bits are tied to 0. See definition under "VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS " Bits[6:0] is only used.
	24:23	**Reserved** Format: MBZ
	22	**Reserved** Format: MBZ
	21	**Forward Gamma Correction Enable** Format: Enable **Programming Notes** Single Pipe IECP Enable must also be set if this is enabled.
	20	**Reserved** Format: MBZ
	19	**Single Pipe Enable** **Description** Indicates that the Capture Pipe features that only exist in a single pipe can be enabled. This bit must be set if any of the following features are enabled: Demosaic Denoise with one of the RGBA input formats IECP only mode with Forward Gamma Correction enabled with RGB input formats (All other modes are not supported in single pipe) This bit must be set if any of the following features are enabled: IECP only mode with either Color Correction Matrix, Forward Gamma Correction, Front-End CSC enabled. Value / Name 1 / Enable 0 / Default **[Default]** **Programming Notes** Note that the pixel throughput is 1/2 when this mode is selected. The **Global IECP Enable** must also be set.

VEBOX_STATE

	18	**Disable Temporal Denoise Filter** If set this bit will force the denoise filter to only use the spatial filter. This will eliminate the read of the previous denoise surface and STMM/Denoise History surface and the write of the current denoised surface and STMM/Denoise History surface. **Programming Notes** **The Global IECP Enable** or **Demosaic Enable** must be set along with this bit. This bit must be set if the input to Denoise is RGB. This bit must not be set if the Deinterlacer is enabled. This bit must be clear if both **DN Enable**=0 and **Hot Pixel Filtering Enable**=0 This bit must be set if **Hot Pixel Filtering Enable**=1 and both DN and DI are disabled
	17	**Disable Encoder Statistics** If set this bit will disable writing the per block Encoder statistics. The memory format is not changed, so the area set aside for these statistics will still be there.
	16	**LACE Correction Enable** This bit enables the correction of the image according to the local ACE LUT tables. This is independent from the enable for the collection of LACE histograms. **Programming Notes** LACE correction is only enabled if both this bit and the **Global IECP Enable** are set. The **ACE Enable** bit should also be set if this bit is set, since ACE correction can be used for part of the luma range instead of LACE.
	15:14	**Single Slice VEBOX Enable** For products that have 2 entire VEBOXes that automatically split the frame, this enable emulates a 1 VEBOX product, running at 1/2 speed and only outputting a single set of per command statistics. **Value** / **Name** 00b — Both Slices Enabled 01b — Slice 0 Enabled 10b — Slice 1 Enables 11b — Reserved **Programming Notes** This field must be set if 00b for products that don't have 2 slices.
	13	**Hot Pixel Filtering Enable** Enables hot pixel detection/filtering.
	12	**Alpha Plane Enable** Enables the reading of an independent Alpha plane. Mutually exclusive with Vignette Enable. If **Alpha from State Select** is set it overrides this bit. **Programming Notes** IECP must also be enabled and output format must have alpha if this bit is enabled. Should be 0 if Alpha from State Select is 1.

VEBOX_STATE

11 Vignette Enable

Enables Vignette Correction surface read and correction in IECP. Mutually exclusive with Alpha Plane Enable.

Programming Notes
Demosaic must also be enabled if this bit is enabled.

10 Demosaic Enable

The Demosaic will be used, and White balance statistics will be gathered. The Capture Pipe State Table will be read. This bit is mutually exclusive with **DI Enable.**

Programming Notes
IECP must also be enabled if this bit is enabled.

9:8 DI Output Frames

Indicates which frames to output in DI mode.

Value	Name
00b	Output Both Frames
01b	Output Previous Frame Only
10b	Output Current Frame Only

Programming Notes
Field is ignored if DI Enable = 0. If Previous Frame Only or Current Frame Only are selected, then the **LACE Single Histogram Set** must not try to collect a histogram from the disabled frame.
Field must be programmed to 10 (Output Current Frame Only) for DI First Frame.

7 444 -> 422 Downsample Method

Value	Name
1	Average horizontally aligned chromas
0	Drop right chroma of the pair **[Default]**

Programming Notes

444->422	422->420	Description
0	0	No averaging, only down sampling
0	1	Not Supported
1	0	Only Horizontal averaging
1	1	Horizontal and Vertical averaging

6 422 -> 420 Downsample Method

Value	Name
1	Average vertically aligned chromas
0	Drop lower chroma of the pair **[Default]**

VEBOX_STATE

Programming Notes
To enable averaging in case of 420 (NV12/P016) output formats, 444->422 and 422->420 should be set.

5 **DN/DI First Frame**

Format:	Enable

Indicates that this is the first frame of the stream, so previous clean is not available.

Value	Name
0	Not first field; previous clean surface state is valid
1	First field; previous clean surface state is invalid

Programming Notes
If both DN and DI are disabled, this bit must be 0.

4 **DI Enable**

Format:	Enable

Deinterlacer is bypassed if this is disabled: the output is the same as the input (same as a 2:2 cadence).

FMD and STMM are not calculated and the values in the response message are 0.

Value	Name
0	Do not calculate DI
1	Calculate DI

3 **DN Enable**

Format:	Enable

Denoise is bypassed if this is low - BNE is still calculated and output, but the denoised fields are not. VDI does not read in the denoised previous frame but uses the pointer for the original previous frame.

Value	Name
0	Do not denoise frame
1	Denoise frame

Programming Notes
If DN and/or Hotpixel are the only functions enabled then the only output is the Denoised Output which is the same surface format as the input. To get a format conversion with DN only, enable the Global IECP bit, but disable all the individual functions. The IECP output uses the output surface format.
If DN is used with RGB then the **Global IECP** Enable must also be

2 **Global IECP Enable**

Indicates if any of the IECP features is enabled.

If this is disabled then no state will be read from any of the state pointers. If set then the IECP state will be read.

VEBOX_STATE

	1	**Color Gamut Compression Enable** Indicates if the Gamut Compression feature is enabled. If set then the Gamut State will be read. VEB_VERTEXTABLE_STATE is only needed if this bit is set.
	0	**Color Gamut Expansion Enable** Indicates if the Gamut Expansion feature is enabled. If set then the Gamut State will be read.
2	31:12	**DN/DI State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the DN/DI State buffer. This points to a buffer containing the 10 Dwords of the DN/DI state.
	11:0	**Reserved** Format: MBZ
3	31:16	**Reserved** Format: MBZ
	15:0	**DN/DI State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the DN/DI State Buffer.
4	31:12	**IECP State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the IECP State buffer. This points to a buffer containing the 64 Dwords of IECP state.
	11:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:0	**IECP State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the IECP State Buffer Table.
6	31:12	**Gamut State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Gamut State buffer. This points to a buffer containing the 42 Dwords of Gamut Compression / Gamut Expansion state.
	11:0	**Reserved** Format: MBZ

VEBOX_STATE

7	31:16	**Reserved** Format: MBZ
	15:0	**Gamut State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Gamut State Buffer.
8	31:12	**Vertex Table State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Vertex Table. This points to a buffer containing the 512 Dwords of the Gamut Compression Vertex Table.
	11:0	**Reserved** Format: MBZ
9	31:16	**Reserved** Format: MBZ
	15:0	**Vertex Table State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Vertex State Buffer.
10	31:12	**Capture Pipe State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Capture Pipe State Table. This points to a buffer containing the X Dwords of the Capture Pipe State.
	11:0	**Reserved** Format: MBZ
11	31:16	**Reserved** Format: MBZ
	15:0	**Capture Pipe State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Capture Pipe State Table.
12	31:12	**LACE LUT Table State Pointer Low** Format: GraphicAddress[31:12] Bits [31:12] of the starting address of the LACE Look-up Tables.
	11:0	**Reserved** Format: MBZ

VEBOX_STATE

13 | 31:30 | Arbitration Priority Control - For LACE LUT

Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
0	Highest priority
1	Second highest priority
2	Third highest priority
3	Lowest priority

29:16 | Reserved

Format:	MBZ

15:0 | LACE LUT Table State Pointer High

Format:	GraphicAddress[47:32]

Bits [47:32] of the starting address of the LACE Look-up Tables.

14..15 | 63:12 | Gamma Correction Values Address

Format:	GraphicsAddress63-12

Specifies the 4K byte aligned address reading the Gamma Correction Values in case enabled.

11:0 | Reserved

Format:	MBZ

VEBOX_SURFACE_STATE

VEBOX_SURFACE_STATE	
Source:	VideoEnhancementCS
Length Bias:	2

Description
The input and output data containers accessed are called "surfaces". Surface state is sent to VEBOX via an inline state command rather than using binding tables. SURFACE_STATE contains the parameters defining each surface to be accessed, including its size, format, and offsets to its subsurfaces. The surface's base address is in the execution command. Despite having multiple input and output surfaces, we limit the number of surface states to one for input surfaces and one for output surfaces. The other surfaces are derived from the input/output surface states.
The Current Frame Input surface uses the Input SURFACE_STATE
The Previous Denoised Input surface uses the Input SURFACE_STATE. (For 16-bit Bayer pattern inputs this will be 16-bit.)
The Current Denoised Output surface uses the Input SURFACE_STATE. (For 16-bit Bayer pattern inputs this will be 16-bit.)
The STMM/Noise History Input surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.
The STMM/Noise History Output surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.
The Current Deinterlaced/IECP Frame Output surface uses the Output SURFACE_STATE.
The Previous Deinterlaced/IECP Frame Output surface uses the Output SURFACE_STATE.
The FMD per block output / per Frame Output surface uses the Linear SURFACE_STATE (see note below).
The Alpha surface uses the Linear A8 SURFACE_STATE with Width/Height equal to Input Surface. Pitch is width rounded to next 64.
The Skin Score surface uses the Output SURFACE_STATE.
The STMM height is the same as the Input Surface height except when the input **Surface Format** is Bayer Pattern and the **Bayer Pattern Offset** is 10 or 11, in which case the height is the input height + 4. For Bayer pattern inputs when the **Bayer Pattern Offset** is 10 or 11, the Current Denoised Output/Previous Denoised Input will also have a height which is the input height + 4. For Bayer pattern inputs only the Current Denoised Output/Previous Denoised Input are in Tile-Y.
The linear surface for FMD statistics is linear (not tiled). The height of the per block statistics is (Input Height +3)/4 - the Input Surface height in pixels is rounded up to the next even 4 and divided by 4. The width of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 16 bytes. The pitch of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 64 bytes.
The STMM surfaces must be identical to the Input surface except for the tiling mode must be Tile-Y and the pitch is specified in DW7. The pitch for the Current Denoised Output/Previous Denoised Input is specified in DW7. The width and height must be a multiple of 4 rounded up from the input height.

VEBOX_SURFACE_STATE

The Vignette Correction surface uses the Linear 16-bit SURFACE_STATE with :
Width=(Ceil(Image Width / 4) +1) * 4
Height= Ceil(Image Height / 4) +1
Pitch in bytes is (vignette width *2) rounded to the next 64

Programming Notes

VEBOX may write to memory between the surface width and the surface pitch for output surfaces.

VEBOX can support a frame level X/Y offset which allows processing of 2 side-by-side frames for certain 3D video formats.

The X/Y Offset for Frame state applies only to the Current Frame Input and the Current Deinterlaced/IECP Frame Output and Previous Deinterlaced/IECP Frame Output. The statistics surfaces, the denoise feedback surfaces and the alpha/vignette surfaces have no X/Y offsets.

For 8bit Alpha input, when converted to 16bit output, the 8 bit alpha value is replicated to both the upper and lower 8 bits to form the 16 bit alpha value.

Skin Score Output Surface uses the same tiling format as the Output surface.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h VEBOX Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h VEBOX Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 7h — Name: DWORD_COUNT_n **[Default]** — Description: (Excludes DWords 0, 1)

VEBOX_SURFACE_STATE

1 — 31:1 — Reserved

Format:	MBZ

1 — 0 — Surface Identification

Specifies which set of surfaces this command refers to:

Value	Name
1	Output surface (all except the Denoised Current output surface)
0	Input surface and Denoised Current Output Surface

2 — 31:18 — Height

Format:	U14

This field specifies the height of the surface in units of pixels. For PLANAR surface formats, this field indicates the height of the Y (luma) plane.

Value	Name	Description	Exists If
[15, 16383]		representing heights [16,16384]	
[15, 8191]			//Scalar Enabled - For Input surface only
[63, 2047]			//Scalar + SFC Enabled - For Input surface only

Programming Notes

Height (field value + 1) must be a multiple of 2 for PLANAR_420 surfaces.
Height (field value +1) must be a multiple of 2 when the deinterlace function is enabled (field mode) or when the denoise function is enabled with **Progressive DN** = 0. It must be a multiple of 4 when interleaved deinterlace/denoise and PLANAR_420 are both being used.
VEBOX supports a minimum height of 16.

Height (field value + 1) must be a multiple of 2 for Bayer surfaces.

2 — 17:4 — Width

Format:	U14

This field specifies the width of the surface in units of pixels. For PLANAR surface formats, this field indicates the width of the Y (luma) plane.

Value	Name	Description	Exists If
[63,16383]		representing widths [64,16384]	
[63,8191]			//Scalar Enabled - For Input surface only
[63,2047]			//Scalar and SFC Enabled - For Input Surface only

Programming Notes

The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the **Surface Pitch** field).

VEBOX_SURFACE_STATE

		Width (field value + 1) must be a multiple of 2 for PLANAR_420, PLANAR_422, and all YCRCB_* surfaces, and must be a multiple of 4 for PLANAR_411 surfaces. **VEBOX** supports a minimum width of 64
	3:0	**Reserved** Format: MBZ
3	31:28	**Surface Format** Format: U4 Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.

Value	Name	Description
0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	NV12 with Interleave Chroma set
5	PACKED_444A_8	IECP input/output only
6	PACKED_422_16	IECP input/output only
7	R10G10B10A2_UNORM / R10G10B10A2_UNORM_SRGB	IECP output only
8	R8G8B8A8_UNORM / R8G8B8A8_UNORM_SRGB	Hot Pixel/Denoise and/or IECP input/output
9	PACKED_444_16	IECP input/output only
10	PLANAR_422_16	IECP input/output only
11	Y8_UNORM	
12	PLANAR_420_16	IECP input/output only
13	R16G16B16A16	Hot Pixel/Denoise and/or IECP input/output
14	Bayer pattern	Demosaic input only
15	Y16_UNORM	Denoise/IECP input/output

	27	**Interleave Chroma** Format: Enable This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.
	26:25	**Bayer Pattern Offset** Specifies the starting pixel offset for the Bayer pattern used for Capture Pipe.

Value	Name
00b	Pixel at X=0, Y=0 is Blue

VEBOX_SURFACE_STATE

01b	Pixel at X=0, Y=0 is Red
10b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Red
11b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Blue

24 Bayer Pattern Format

Specifies the format of the Bayer Pattern:

Value	Name
0b	8-bit input at a 8-bit stride
1b	16-bit input at a 16-bit stride

23:21 Reserved

Format:	MBZ

20 Reserved

Format:	MBZ

19:3 Surface Pitch

Format:	U17 pitch in (Bytes - 1)

This field specifies the surface pitch in (#Bytes - 1):

Value	Name	Description
[63, 131071]	For other linear surfaces	[64B, 128KB]
[511, 131071]	For X-tiled surface	[512B, 128KB] = [1tile, 256 tiles]
[127, 131071]	For Y-tiled surfaces	[128B,128KB] = [1 tile, 1024 tiles]

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile width. For linear surfaces, the pitch must be a multiple of 64. If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.

2 Half Pitch for Chroma

Format:	Enable

This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the **Surface Pitch** field. This field is only used for PLANAR surface formats.

Programming Notes
Must be programmed to Zero always as this field is not used

1 Tiled Surface

Format:	Boolean

This field specifies whether the surface is tiled.

Value	Name	Description
1	True	Tiled
0	False	Linear

VEBOX_SURFACE_STATE

		Programming Notes Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory. The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
	0	**Tile Walk** Format: 3D_TileWalk This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions. This field is ignored when the surface is linear. **Value** / **Name** 0 TILEWALK_XMAJOR 1 TILEWALK_YMAJOR **Programming Notes** The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
4	31:29	**Reserved** Format: MBZ
	28:16	**X Offset for U** Format: U13 Pixel Offset This field must be zero for the VEBOX surface formats
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for U** Format: U15 Row Offset This field specifies the veritical offset in rows from the start (origin) or the Luma(Y) plane to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled. This field is only used for PLANAR surface formats. **Programming Notes** This field must indicate an even number (bit 0 = 0). This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line) For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for U should be an integral multiple of the Tile height of the Luma plane
5	31:29	**Reserved** Format: MBZ

VEBOX_SURFACE_STATE

	28:16	**X Offset for V** Format: U13 Pixel Offset This field must be zero for the VEBOX surface formats.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for V** Format: U15 Row Offset This field specifies the veritical offset in rows from the start (origin) of the Luma(Y) plan**e** to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with **Interleave Chroma** disabled. **Programming Notes** This field must indicate an even number (bit 0 = 0). This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line). For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for V should be an integral multiple of the Tile height of the Luma plane
6	31	**Reserved** Format: MBZ
	30:16	**X Offset for Frame** Format: U15 Pixel Offset This is an offset in X from the Surface Base Address in pixels for all planes using this surface. For U/V planes this is added to the X Offset for U/V. After converting to bytes this must be an integer multiple of cache lines in the tiling mode. This specifies the edge of the frame, so adjacent pixels needed for Denoise/Deinterlace/Demosaic are replicated or mirrored. **Programming Notes** If**Y Offset for Frame** >0 the X Offset must be 0. If memory compression is enabled then this must be an even number of cache lines.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for Frame** Format: U15 Pixel Offset This is an offset in Y from the Surface Base Address in pixels for all planes using this surface. For U/V planes this is added to the Y Offset for U/V. After converting to bytes this must be an integer multiple of cache lines in the tiling mode. This specifies the edge of the frame, so adjacent pixels needed for Denoise/Deinterlace/Demosaic are replicated or mirrored. **Programming Notes** If **X Offset for Frame** >0 the Y Offset must be 0. For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for Frame should be an integral multiple of the Tile height.

VEBOX_SURFACE_STATE

7 — 31:17 — Reserved

Format:	MBZ

7 — 16:0 — Derived Surface Pitch

Format:	U17 pitch in (Bytes - 1)

This field specifies the surface pitch in (#Bytes - 1) for the derived surfaces:
STMM/Denoise statistic surface is described when the **Surface Identification** bit is 0 (Input Surface).
The (Current Denoise Output)/(Previous Denoise Input) surfaces are described when the bit is 1 (Output Surface).

Value	Name	Description	Exists If
[63, 131071]		[64B, 128KB]	[Tiled Surface] == 0
[511, 131071]		[512B, 128KB] = [1tile, 256 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 0)
[127, 131071]		[128B,128KB] = [1 tile, 1024 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 1)

Programming Notes
In DN Only mode, the pitch for the (Current Denoise Output)/(Previous Denoise Input) and the Surface Pitch must be programed the same.
The pitch must be a multiple of the tile width.

8 — 31:17 — Reserved

Format:	MBZ

8 — 16:0 — Surface Pitch for Skin Score Output Surfaces

Format:	U17 pitch in (Bytes - 1)

This field specifies the surface pitch in (#Bytes - 1) for the Skin Score Output surface if enabled;
This is present only in the output surface format and reserved for Input surface format. The height and width are the same as in the Output surface mentioned above.

Value	Name	Description	Exists If
[63, 131071]		[64B, 128KB]	[Tiled Surface] == 0
[511, 131071]		[512B, 128KB] = [1tile, 256 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 0)
[127, 131071]		[128B,128KB] = [1 tile, 1024 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 1)

Programming Notes
The pitch must be a multiple of the tile width.

VEBOX_TILING_CONVERT

VEBOX_TILING_CONVERT

Source: VideoEnhancementCS

Length Bias: 2

This command takes the input surface and writes directly to the output surface at high speed. The surface format and width/height of the input and output must be the same, only the tiling mode and pitch can change.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 3h / / (Excludes DWords 0, 1)
1..2	63:12	**Input Address** Format: GraphicsAddress63-12 Specifies bits 47:12 of the 4Kbyte-aligned frame buffer address for reading the current frame.
	11	**Reserved** Format: MBZ

VEBOX_TILING_CONVERT		
	10:0	**Input Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
3..4	63:12	**Output Address** Format: GraphicsAddress63-12 Specifies bits 47:12 of the 4Kbyte-aligned frame buffer address for writing the current frame.
	11	**Reserved** Format: MBZ
	10:0	**Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS

Wait Notification

wait - Wait Notification

Source: Eulsa

Length Bias: 4

The wait instruction evaluates the value of the notification count register nreg. If nreg is zero, thread execution is suspended and the thread is put in 'wait_for_notification' state. If nreg is not zero (i.e., one or more notifications have been received), nreg is decremented by one and the thread continues executing on the next instruction. If a thread is in the 'wait_for_notification' state, when a notification arrives, the notification count register is incremented by one. As the notification count register becomes nonzero, the thread wakes up to continue execution and at the same time the notification register is decremented by one. If only one notification arrived, the notification register value becomes zero. However, during the above mentioned time period, it is possible that more notifications may arrive, making the notification register nonzero again.

When multiple notifications are received, software must use wait instructions to decrement notification count registers for each notification.

Notification register n0.0:ud is for thread to thread communication (via the Message Gateway shared function) and n0.1:ud for host to thread communication (through MMIO registers). See the Message Gateway chapter for thread-thread communication for host-to-thread communication.

Format:

wait (exec_size) nreg

Restriction
Restriction : src0 and dst must be n0.0, n0.1, or n0.2.
Restriction : Execution size must be 1 as the notification registers are scalar.
Restriction : Predication is not allowed.
Restriction : Two back-to-back wait instructions are not allowed. At minimum, a nop instruction must be inserted between two wait instructions

Syntax
`wait (1) n#`

Pseudocode
`N/A`

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0	127:64	**Sources**

wait - Wait Notification

		Exists If: ([Operand Control][Src1.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG**
	127:64	**Sources** Exists If: ([Operand Control][Src1.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_IMM32**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

While

while - While

Source: Eulsa

Length Bias: 4

The while instruction marks the end of a do-while block. The instruction first evaluates the loop termination condition for each channel based on the current channel enables and the predication flags specified in the instruction. If any channel has not terminated, a branch is taken to a destination address specified in the instruction, and the loop continues for those channels. Otherwise, execution continues to the next instruction.ld point to the first instruction with the do label of the do-while block of code. It should be a negative number for the backward referencing. In GEN binary, JIP is at location dst and must be of type W (signed word integer).
If SPF is ON, none of the PcIP are updated.

Format:
[(pred)] while (exec_size) JIP

Restriction

Restriction : The execution size must be the same for the while instruction and any break and cont instructions of the same code block.

Syntax

```
[(pred)] while (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if (WrEn.chan[n] ) { PcIP[n] = IP +
JIP; } else { PcIP[n] = IP + 1; } } if ( | PMask == 1 ) { // any enabled channel
true Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: SrcType
	90:89	**Src1.RegFile** Format: RegFile

while - While

	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

XY_COLOR_BLT

XY_COLOR_BLT

Source: BlitterCS

Length Bias: 2

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.
This instruction is optimized to run at the maximum memory write bandwidth.
The typical (and fastest) Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 50h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name 1xb: Write Alpha Channel x1b: Write RGB Channel
	19:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value / Name / Description 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled: Tile-X or Tile-Y
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 05h Format: =n
1 BR13	31	**Reserved** Format: MBZ

XY_COLOR_BLT

	30	**Clipping Enabled** Value: 0b — Disabled Value: 1b — Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b — 8 Bit Color 01b — 16 Bit Color(565) 10b — 16 Bit Color(1555) 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).
6 BR16	31:0	**Solid Pattern Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_FAST_COPY_BLT

XY_FAST_COPY_BLT

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a color source copy where the only operands involved are a color source and destination of the same bit width. The source and destination surfaces CANNOT overlap. The hardware assumes this whenever this Fast_Copy command is given to it. For overlapping Blits, use the traditional XY_SRC_COPY_BLT command (for overlap determination, read the description given in the XY_SRC_COPY_BLT command). Note that this command does not support Clipping operations. This new blit command will happen in large numbers, consecutively, possibly an entire batch will comprise only new blit commands Legacy commands and new blit command will not be interspersed. If they are, they will be separated by implied HW flush: Whenever there is a transition between this new Fast Blit command and the Legacy Blit commands, the HW will impose an automatic flush BEFORE the execution (at the beginning) of the next blitter command. New blit command can use any combination of memory surface type - linear, tiledX, tiledY, and the tiling information is conveyed as part of the new Fast Copy command. The Fast Copy Blit supports the new 64KB Tiling defineds.The starting pixel of Fast Copy blit for both source and destination should be on an OWord boundary.

Note that when two sequential fast copy blits have different source surfaces, but their destinations refer to the same destination surfaces and therefore destinations overlap it is imperative that a Flush be inserted between the two blits.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 42h Format: Opcode
	21:20	**Source Tiling Method** SW is required to flush the HW before changing the polarity of these bits for subsequent blits. Value / Name / Description 00b / Linear (Tiling Disabled) / 01b / Legacy Tile-X / 10b / Tile-Y / Choosing between 'Legacy Tile-Y' or the 'New 4K Tile-YF' can be done in DWord 1, Bit[31]. 11b / 64kb Tiling /
	19:15	**Reserved** Format: MBZ

XY_FAST_COPY_BLT

	14:13	**Destination Tiling Method** SW is required to flush the HW before changing the polarity of these bits for subsequent blits.

Value	Name	Description
00b	Linear (Tiling Disabled)	
01b	Legacy Tile-X	
10b	Tile-Y	Choosing between 'Legacy Tile-Y' or the 'New 4K Tile-YF' can be done in DWord 1, Bit[30].
11b	64kb Tiling	

	12:8	**Reserved**

Format:	MBZ

	7:0	**DWord Length**

Default Value:	08h Excludes DWORD 0,1
Format:	=n

08h

1 BR13	31	**Tile Y Type for Source** Source being Tile-Y can be selected in DWord 0, Bit[21:20].

Value	Name
0b	Legacy Tile-Y
1b	New 4k Tile-YF

	30	**Tile Y Type for Destination** Destination being Tile-Y can be selected in DWord 0, Bit[14:13].

Value	Name
0b	Legacy Tile-Y
1b	New 4k Tile-YF

	29:27	**Reserved**

Format:	MBZ

	26:24	**Color Depth**

Value	Name
000b	8 bit color
001b	16 bit color (565)
011b	32 bit color
100b	64 bit color (for 64KB Tiling)
101b	128 bit color (for 64KB Tiling)

	23:16	**Reserved**

Format:	MBZ

XY_FAST_COPY_BLT

	15:0	**Destination Pitch (signed)** 2's Complement. For Fast Copy Blits the pitch cannot be a negative number. As a result the pitch must be a positive value and bit 15 of the destination pitch must be zero. For **Linear surfaces**, the pitch has to be an **OWord (16byte) multiple** for Linear-Linear copies; and **CacheLine aligned** for Tiled-Linear type copies. The value should be specified as a number in *bytes*. For **Tiled surfaces**, the pitch has to be a **multiple of the Tile width** (X direction width of the Tile). This means the pitch value will always be **Cache Line aligned (64byte multiple**) and the number or value mentioned in this field here should be specified as a number of Dwords (4byte quantity).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16-bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16-bit signed number. The start pixel for Fast Copy blit should be on an OWord boundary.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16-bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16-bit signed number.
4..5	63:0	**Destination Base Address** Format: **GraphicsAddress63-0** This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16-bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16-bit signed number. The start pixel for Fast Copy blit should be on an OWord boundary.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned)** 2's Complement. For Fast Copy Blits the pitch cannot be a negative number. As a result the pitch must be a positive value and bit 15 of the source pitch must be zero. For **Linear surfaces**, the pitch has to be an **OWord (16byte) multiple** for Linear-Linear copies; and **CacheLine aligned** for Linear-Tiled type copies. The value , and the value should be specified as a number in *bytes*. For **Tiled surfaces**, the pitch has to be a **multiple of the Tile width** (X direction width of the Tile). This means the pitch value will always be **Cache Line aligned (64byte multiple**) and the number or value mentioned in this field here should be specified as a number of Dwords (4byte quantity).
8..9	63:0	**Source Address** Format: **GraphicsAddress63-0** This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).

XY_FULL_BLT

XY_FULL_BLT

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and pattern operands are the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 55h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_FULL_BLT

	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Dest Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 0Ah
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value / Name 0b / Disabled 1b / Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value / Name 00b / 8 Bit Color 01b / 16 Bit Color(565) 10b / 16 Bit Color(1555) 11b / 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_FULL_BLT		
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.
6 BR11	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8..9	63:0	**Source Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.
10..11	63:0	**Pattern Base Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.

XY_FULL_IMMEDIATE_PATTERN_BLT

XY_FULL_IMMEDIATE_PATTERN_BLT

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and immediate pattern operands are the same bit width as the destination operand. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64 DWs) for 8, 16, and 32 bpp color patterns. DWL indicates the total number of Dwords of immediate data.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 74h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** 0b Tiling Disabled (Linear) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_FULL_IMMEDIATE_PATTERN_BLT

	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable** Value / Name / Description: 0b — Tiling Disabled (Linear Blit) 1b — Tiling Enabled — : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1 08 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value / Name: 0b — Disabled 1b — Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value / Name: 00b — 8 Bit Color 01b — 16 Bit Color(565) 10b — 16 Bit Color(1555) 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_FULL_IMMEDIATE_PATTERN_BLT		
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
6 BR11	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8..9	63:0	**Source Address** Format: GraphicsAddress63-0 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
10..n	31:0	**Immediate Data 0**

XY_FULL_MONO_PATTERN_BLT

XY_FULL_MONO_PATTERN_BLT

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The pattern operand is monochrome and the source operand is the same bit width as the destination operand.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 57h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved**

XY_FULL_MONO_PATTERN_BLT

	15	**Src Tiling Enable** | Value | Name | Description | | 0b | Tiling Disabled (Linear Blit) | | | 1b | Tiling Enabled | : Tile-X or Tile-Y. |
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable** | Value | Name | Description | | 0b | Tiling Disabled (Linear Blit) | | | 1b | Tiling Enabled | : Tile-X or Tile-Y. |
	10:8	**Pattern Vectical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** | Value | Name | | 0Ch | |
1 BR13	31	**Solid Pattern Select** | Value | Name | | 0 | No Solid Pattern | | 1 | Solid Pattern |
	30	**Clipping Enabled** | Value | Name | | 0b | Disabled | | 1b | Enabled |
	29	**Reserved** | Format: | MBZ |
	28:27	**Mono Source Transparency Mode** | Value | Name | | 0 | Use Background | | 1 | Transparency Enabled |
	26	**Reserved** | Format: | MBZ |
	25:24	**Color Depth** | Value | Name | | 00b | 8 Bit Color | | 01b | 16 Bit Color(565) | | 10b | 16 Bit Color(1555) |

XY_FULL_MONO_PATTERN_BLT

		11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
6 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8..9	63:0	**Source Address** Format: GraphicsAddress63-0 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
10 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
11 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
12 BR20	31:0	**Pattern Data 0** (least significant DW)

XY_FULL_MONO_PATTERN_BLT		
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

Source: BlitterCS

Length Bias: 2

The full BLT provides the ability to specify all 3 operands: destination, source, and pattern. The pattern and source operands are monochrome.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.
All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation. The monochrome source transparency mode works identical to the pattern transparency mode.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 58h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable** Value: 0b — Name: Tiling Disabled (Linear Blit) Value: 1b — Name: Tiling Enabled — Description: : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y = 0.
	7:0	**DWord Length** Value: 0Ch — Name:
1 BR13	31	**Solid Pattern Select** Value: 0 — Name: No Solid Pattern Value: 1 — Name: Solid Pattern
	30	**Clipping Enabled** Value: 0b — Name: Disabled Value: 1b — Name: Enabled
	29	**Mono Source Transparency Mode** Value: 0 — Name: Use Background Value: 1 — Name: Transparency Enabled
	28	**Mono Pattern Transparency Mode** Value: 0 — Name: Use Background Value: 1 — Name: Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b — Name: 8 Bit Color

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

		01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 This bitfield contains the base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: GraphicsAddress63-0
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
11 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
12 BR20	31:0	**Pattern Data 0** (least significant DW)
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_SRC_BLT

XY_FULL_MONO_SRC_BLT

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is monochrome and the pattern operand is the same bit width as the destination.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.

All non-text and non-immediate monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the Destination X1 coordinate.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Negative Stride (= Pitch) is NOT ALLOWED

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 56h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved** Format: MBZ

XY_FULL_MONO_SRC_BLT

	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y = 0.
	7:0	**DWord Length** Value / Name 0Ah /
1 BR13	31	**Reserved** Format: / MBZ
	30	**Clipping Enabled** Value / Name 0b / Disabled 1b / Enabled
	29	**Mono Source Transparency Mode** Value / Name 0 / Use Background 1 / Transparency Enabled
	28:26	**Reserved** Format: / MBZ
	25:24	**Color Depth** Value / Name 00b / 8 Bit Color 01b / 16 Bit Color(565) 10b / 16 Bit Color(1555) 11b / 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).

XY_FULL_MONO_SRC_BLT		
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: GraphicsAddress63-0
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10..11 28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The programmed Pattern Base Address must always be Cache Line (64byte) aligned.	63:0	**Pattern Base Address** Format: GraphicsAddress63-0

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is a monochrome and the immediate pattern operand is the same bit width as the destination. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns. The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation. The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8. Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 75h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1 08 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b Disabled 1b Enabled
	29	**Mono Source Transparency Mode** 0 Use Background 1 Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b 8 Bit Color 01b 16 Bit Color(565) 10b 16 Bit Color(1555) 11b 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT		
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: GraphicsAddress63-0
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10..n	31:0	**Immediate Data**

XY_MONO_PAT_BLT

XY_MONO_PAT_BLT

Source: BlitterCS

Length Bias: 2

MONO_PAT_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is loaded from the instruction stream.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 52h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_MONO_PAT_BLT

	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Value: 08h Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Reserved** Format: MBZ
	28	**Mono Pattern Transparency Mode** 0: Use Background 1: Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.

XY_MONO_PAT_BLT		
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 BR20	31:0	**Pattern Data 0**
9 BR21	31:0	**Pattern Data 1**

XY_MONO_PAT_FIXED_BLT

XY_MONO_PAT_FIXED_BLT

Source: BlitterCS

Length Bias: 2

MONO_PAT_FIXED_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is one of 10 fixed patterns described below. The pattern seeds can still be used with the fixed patterns, creating even more fixed patterns. This eliminates 2 doublewords compared to the XY_MONO_PAT_BLT command packet.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 59h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19	**Reserved** Format: MBZ
	18:15	**Fixed Pattern** 0000b HS_HORIZONTAL 0001b HS_VERTICAL 0010b HS_FDIAGONAL 0011b HS_BDIAGONAL

XY_MONO_PAT_FIXED_BLT

		0100b: HS_CROSS 0101b: HS_DIAGCROSS 0110b: Reserved 0111b: Reserved 1000b: Screen Door 1001b: SD Wide 1010b: Walking Bit (one) 1011b: Walking Zero 1100b: Reserved 1101b: Reserved 1110b: Reserved 1111b: Reserved
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Format: =n Value: 06h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Reserved** Format: MBZ

XY_MONO_PAT_FIXED_BLT

	28	**Mono Pattern Transparency Mode** Value: 0 — Use Background Value: 1 — Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b — 8 Bit Color Value: 01b — 16 Bit Color(565) Value: 10b — 16 Bit Color(1555) Value: 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
6 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_MONO_SRC_COPY_BLT

XY_MONO_SRC_COPY_BLT

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a monochrome source copy where the only operands involved is a monochrome source and destination. The source and destination operands cannot overlap therefore the X and Y directions are always forward.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation. Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 54h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_MONO_SRC_COPY_BLT

	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Value: 08h Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Mono Source Transparency Mode** 0: Use Background 1: Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_MONO_SRC_COPY_BLT		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: **GraphicsAddress63-0**
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: **GraphicsAddress63-0**
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_MONO_SRC_COPY_IMMEDIATE_BLT

XY_MONO_SRC_COPY_IMMEDIATE_BLT

Source: BlitterCS

Length Bias: 2

This instruction allows the Driver to send monochrome data through the instruction stream, eliminating the read latency of the source during command execution.
The IMMEDIATE_BLT data MUST transfer an even number of doublewords and the exact number of quadwords. DWL indicates the total number of Dwords of immediate data.
All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.
The monochrome source data supplied corresponds to the Destination X1 and Y1 coordinates.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 71h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b Tiling Disabled (Linear) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_MONO_SRC_COPY_IMMEDIATE_BLT

	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 06h Excludes DWORD 0,1 06 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b Disabled 1b Enabled
	29	**Mono Source Transparency Mode** 0b Transparency Enabled 1b Use Background
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b 8 Bit Color 01b 16 Bit Color(565) 10b 16 Bit Color(1555) 11b 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_MONO_SRC_COPY_IMMEDIATE_BLT		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: **GraphicsAddress63-0**
6 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8..n	31:0	**Immediate Data**

XY_PAT_BLT

XY_PAT_BLT

Source: BlitterCS

Length Bias: 2

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
If clipping is enabled, all scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 51h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 06h

XY_PAT_BLT

1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0

XY_PAT_BLT		
6..7 28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The programmed Pattern Base Address must always be Cache Line (64byte) aligned.	63:0	**Pattern Base Address** Format: GraphicsAddress63-0

XY_PAT_BLT_IMMEDIATE

XY_PAT_BLT_IMMEDIATE

Source: BlitterCS

Length Bias: 2

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 72h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length**

XY_PAT_BLT_IMMEDIATE

		Default Value: 04h Excludes DWORD 0,1 04 + DWL = (Number of Immediate double)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6..n	31:0	**Immediate Data**

XY_PAT_CHROMA_BLT

XY_PAT_CHROMA_BLT

Source: BlitterCS

Length Bias: 2

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 76h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1

XY_PAT_CHROMA_BLT

1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5	63:0	**Destination Base Address** Format: GraphicsAddress63-0 This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).
6..7	63:0	**Pattern Base Address** Format: GraphicsAddress63-0 (28:06 are implemented) (Note no NPO2 change here). The pattern data must be located in linear memory. The Pattern Base Address must always be Cache Line (64byte) aligned.
8 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
9 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_PAT_CHROMA_BLT_IMMEDIATE

XY_PAT_CHROMA_BLT_IMMEDIATE

Source: BlitterCS

Length Bias: 2

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 77h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.

XY_PAT_CHROMA_BLT_IMMEDIATE

	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 06h Excludes DWORD 0,1 06 + DWL = (Number of Immediate double)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value: 0b – Disabled Value: 1b – Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b – 8 Bit Color Value: 01b – 16 Bit Color(565) Value: 10b – 16 Bit Color(1555) Value: 11b – 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_PAT_CHROMA_BLT_IMMEDIATE		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: **GraphicsAddress63-0**
6 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
7 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)
8..n	31:0	**Immediate Data**

XY_PIXEL_BLT

XY_PIXEL_BLT

Source: BlitterCS

Length Bias: 2

The Destination X coordinate and Destination Y coordinate is compared with the ClipRect registers. If it is within all 4 comparisons, then the pixel supplied in the XY_SETUP_BLT instruction is written with the raster operation to (Destination Y Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

Negative Stride (= Pitch) specified in the Setup command is Not Allowed

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 24h Format: Opcode
	21:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 00h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.

XY_SCANLINES_BLT

XY_SCANLINES_BLT

Source: BlitterCS

Length Bias: 2

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Solid pattern should use the XY_SETUP_MONO_PATTERN_SL_BLT instruction.
ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 25h Format: Opcode
	21:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 01h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_SETUP_BLT

XY_SETUP_BLT

Source: BlitterCS

Length Bias: 2

This setup instruction supplies common setup information including clipping coordinates used by the XY commands: XY_PIXEL_BLT, XY_SCANLINE_BLT, XY_TEXT_BLT, and XY_TEXT_BLT_IMMEDIATE.
These are the only instructions that require that state be saved between instructions other than the Clipping parameters. There are 5 dedicated registers to contain the state for the 3 setup BLT instructions (XY_SETUP_BLT, XY_SETUP_MONO_PATTERN_SL_BLT, and XY_SETUP_CLIP_BLT. All other BLTs use a temporary version of these. The 5 double word registers are: DW1 (Setup Control), DW6 (Setup Foreground color), DW5 (Setup Background color), DW7 (Setup Pattern address), and DW4 (Setup Destination Base Address).

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 01h Format: Opcode
	21:20	**32 bpp Byte Mask** Value: 1xb — Name: Write Alpha Channel Value: x1b — Name: Write RGB Channel
	19:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value: 0b — Name: Tiling Disabled (Linear Blit) Value: 1b — Name: Tiling Enabled (Tile-X or Tile-Y)
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Reserved** Format: MBZ

XY_SETUP_BLT

	30	**Clipping Enabled** Value: 0b — Disabled Value: 1b — Enabled
	29	**Mono Source Transparency Mode** Value: 0b — Use Background Value: 1b — Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b — 8 Bit Color Value: 01b — 16 Bit Color(565) Value: 10b — 16 Bit Color(1555) Value: 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
3 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6 BR05	31:0	**Setup Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All

XY_SETUP_BLT		
7 BR06	31:0	**Setup Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)
8..9 28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The Setup Pattern Base Address for Color Pattern must always be Cache Line (64byte) aligned.	63:0	**Setup Pattern Base Address for Color Pattern** Format: GraphicsAddress63-0

XY_SETUP_CLIP_BLT

XY_SETUP_CLIP_BLT

Source: BlitterCS

Length Bias: 2

This command is used to only change the clip coordinate registers. These are the same clipping registers as the Setup clipping registers above.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 03h Format: Opcode
	21:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value 0b: Tiling Disabled (Linear Blit) Value 1b: Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 01h
1 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
2 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)

XY_SETUP_MONO_PATTERN_SL_BLT

XY_SETUP_MONO_PATTERN_SL_BLT

Source: BlitterCS

Length Bias: 2

This setup instruction supplies common setup information including clipping coordinates used exclusively with the following instruction: XY_SCANLINE_BLT (SLB) - 1 scan line of monochrome pattern and destination are the only operands allowed.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 11h Format: Opcode
	21:20	**32 bpp Byte Mask** 1xb: Write Alpha Channel x1b: Write RGB Channel
	19:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Solid Pattern Select** (SLB and Pixel only) 0: No Solid Pattern 1: Solid Pattern

XY_SETUP_MONO_PATTERN_SL_BLT

	30	**Clipping Enabled** Value: 0b = Disabled; 1b = Enabled
	29	**Reserved** Format: MBZ
	28	**Mono Pattern Transparency Mode** Value: 0b = Use Background; 1b = Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b = 8 Bit Color; 01b = 16 Bit Color(565); 10b = 16 Bit Color(1555); 11b = 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
3 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)

XY_SETUP_MONO_PATTERN_SL_BLT		
4..5 This bitfield contains the base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Setup Destination Base Address** Format: GraphicsAddress63-0
6 BR05	31:0	**Setup Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
7 BR06	31:0	**Setup Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)
8 BR20	31:0	**DW0 (least significant) for a Monochrome Pattern**
9 BR21	31:0	**DW1 (most significant) for a Monochrome Pattern**

XY_SRC_COPY_BLT

XY_SRC_COPY_BLT

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 53h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear) 1b Tiling Enabled : Tile-X or Tile-Y.
	14:12	**Reserved** Format: MBZ

XY_SRC_COPY_BLT

	11	**Dest Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 08h / Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.

XY_SRC_COPY_BLT		
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: **GraphicsAddress63-0**
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
8..9 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Source Base Address** Format: GraphicsAddres63-0

XY_SRC_COPY_CHROMA_BLT

XY_SRC_COPY_CHROMA_BLT

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a color source copy with chroma-keying where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 73h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key)
	16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** 0b Tiling Disabled (Linear) 1b Tiling Enabled : Tile-X or Tile-Y.
	14:12	**Reserved** Format: MBZ

XY_SRC_COPY_CHROMA_BLT

	11	**Dest Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / : Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Value: 0Ah
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_SRC_COPY_CHROMA_BLT		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: GraphicsAddress63-0
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
8..9 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Source Base Address** Format: GraphicsAddress63-0
10 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
11 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_TEXT_BLT

XY_TEXT_BLT

Source: BlitterCS

Length Bias: 2

All source scan lines and pixels that fall within the ClipRect Y and X coordinates are written. The source address corresponds to Destination X1 and Y1 coordinate.

Text is either bit or byte packed. Bit packed means that the next scan line starts 1 pixel after the end of the current scan line with no bit padding. Byte packed means that the next scan line starts on the first bit of the next byte boundary after the last bit of the current line.

Source expansion color registers are always in the SETUP_BLT.

Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 26h Format: Opcode
	21:17	**Reserved** Format: MBZ
	16	**Bit / Byte Packed** Byte packed is for the NT driver. Value: 0 — Name: Bit Value: 1 — Name: Byte
	15:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value: 0b — Name: Tiling Disabled (Linear Blit) Value: 1b — Name: Tiling Enabled — Description: : Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 03h

XY_TEXT_BLT		
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3..4 Address of the first byte on a scan line corresponding to source X1, Y1. Note no NPO2 change here. The source address must always be Cache Line (64byte) aligned.	63:0	**Source Address** Format: GraphicsAddress63-0

XY_TEXT_IMMEDIATE_BLT

XY_TEXT_IMMEDIATE_BLT

Source: BlitterCS

Length Bias: 2

This instruction allows the Driver to send data through the instruction stream that eliminates the read latency of reading a source from memory.

If an operand is in system cacheable memory and either small or only accessed once, it can be copied directly to the instruction stream versus to graphics accessible memory. The IMMEDIATE_BLT data MUST transfer an even number of doublewords.

The BLT engine will hang if it does not get an even number of doublewords. All source scan lines and pixels that fall within the ClipRect X and Y coordinates are written. The source data corresponds to Destination X1 and Y1 coordinate.

Source expansion color registers are always in the SETUP_BLT. NEGATIVE STRIDE (= PITCH) IS NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 31h Format: Opcode
	21:17	**Reserved** Format: MBZ
	16	**Bit / Byte Packed** Byte packed is for the NT driver. Value 0: Bit Value 1: Byte
	15:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ

XY_TEXT_IMMEDIATE_BLT		
	7:0	**DWord Length** Default Value: 01h Excludes DWORD 0,1 01 + DWL = (Number of Immediate double words)h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3..n	31:0	**Immediate Data**

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 2b: Command Reference: Enumerations

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

3D_Color_Buffer_Blend_Factor 1
3D_Color_Buffer_Blend_Function 2
3D_Compare_Function 3
3D_Logic_Op_Function 4
3D_Prim_Topo_Type 5
3D_Stencil_Operation 7
3D_Vertex_Component_Control 8
AddrMode 9
Attribute_Component_Format 10
ChanEn 11
ChanSel 12
COMPONENT_ENABLES 13
CondModifier 14
DepCtrl 15
DstType 16
EU_OPCODE 17
ExecSize 20
FC 21
Fixed Function ID 22
HorzStride 23
Performance Counter Report Formats 24
PredCtrl 25
QtrCtrl 26
RegFile 28
RepCtrl 29
SFID 30
Shader Channel Select 32
Shared Function ID 33
SIMD Mode 34
SrcImmType 35
SrcIndex 36

SrcMod 38
SrcType 39
SURFACE_FORMAT 40
Texture Coordinate Mode 47
ThreadCtrl 48
VertStride 49
Width 50
WRAP_SHORTEST_ENABLE 51

3D_Color_Buffer_Blend_Factor

3D_Color_Buffer_Blend_Factor	
Source:	BSpec
Size (in bits):	5

Value	Name
00h	Reserved
01h	BLENDFACTOR_ONE
02h	BLENDFACTOR_SRC_COLOR
03h	BLENDFACTOR_SRC_ALPHA
04h	BLENDFACTOR_DST_ALPHA
05h	BLENDFACTOR_DST_COLOR
06h	BLENDFACTOR_SRC_ALPHA_SATURATE
07h	BLENDFACTOR_CONST_COLOR
08h	BLENDFACTOR_CONST_ALPHA
09h	BLENDFACTOR_SRC1_COLOR
0Ah	BLENDFACTOR_SRC1_ALPHA
0Bh-10h	Reserved
11h	BLENDFACTOR_ZERO
12h	BLENDFACTOR_INV_SRC_COLOR
13h	BLENDFACTOR_INV_SRC_ALPHA
14h	BLENDFACTOR_INV_DST_ALPHA
15h	BLENDFACTOR_INV_DST_COLOR
16h	Reserved
17h	BLENDFACTOR_INV_CONST_COLOR
18h	BLENDFACTOR_INV_CONST_ALPHA
19h	BLENDFACTOR_INV_SRC1_COLOR
1Ah	BLENDFACTOR_INV_SRC1_ALPHA

3D_Color_Buffer_Blend_Function

3D_Color_Buffer_Blend_Function		
Source:	BSpec	
Size (in bits):	3	
Value	**Name**	**Description**
0	BLENDFUNCTION_ADD	BLENDFUNCTION_ADD
1	BLENDFUNCTION_SUBTRACT	BLENDFUNCTION_SUBTRACT
2	BLENDFUNCTION_REVERSE_SUBTRACT	BLENDFUNCTION_REVERSE_SUBTRACT
3	BLENDFUNCTION_MIN	BLENDFUNCTION_MIN
4	BLENDFUNCTION_MAX	BLENDFUNCTION_MAX
5 - 7	Reserved	

3D_Compare_Function

3D_Compare_Function		
Source:	BSpec	
Size (in bits):	3	
Value	**Name**	**Description**
0h	COMPAREFUNCTION_ALWAYS	Always pass
1h	COMPAREFUNCTION_NEVER	Never pass
2h	COMPAREFUNCTION_LESS	Pass if the value is less than the reference
3h	COMPAREFUNCTION_EQUAL	Pass if the value is equal to the reference
4h	COMPAREFUNCTION_LEQUAL	Pass if the value is less than or equal to the reference
5h	COMPAREFUNCTION_GREATER	Pass if the value is greater than the reference
6h	COMPAREFUNCTION_NOTEQUAL	Pass if the value is not equal to the reference
7h	COMPAREFUNCTION_GEQUAL	Pass if the value is greater than or equal to the reference

3D_Logic_Op_Function

3D_Logic_Op_Function		
Source:	BSpec	
Size (in bits):	4	
Value	**Name**	**Description**
0h	LOGICOP_CLEAR	BLACK; all 0's
1h	LOGICOP_NOR	NOTMERGEPEN; NOT (S OR D)
2h	LOGICOP_AND_INVERTED	MASKNOTPEN; (NOT S) AND D
3h	LOGICOP_COPY_INVERTED	NOTCOPYPEN; NOT S
4h	LOGICOP_AND_REVERSE	MASKPENNOT; S AND NOT D
5h	LOGICOP_INVERT	NOT; NOT D
6h	LOGICOP_XOR	XORPEN; S XOR D
7h	LOGICOP_NAND	NOTMASKPEN; NOT (S AND D)
8h	LOGICOP_AND	MASKPEN; S AND D
9h	LOGICOP_EQUIV	NOTXORPEN; NOT (S XOR D)
Ah	LOGICOP_NOOP	NOP; D
Bh	LOGICOP_OR_INVERTED	MERGENOTPEN; (NOT S) OR D
Ch	LOGICOP_COPY	COPYPEN; S
Dh	LOGICOP_OR_REVERSE	MERGEPENNOT; S OR NOT D
Eh	LOGICOP_OR	MERGEPEN; S OR D
Fh	LOGICOP_SET	WHITE; all 1's

3D_Prim_Topo_Type

3D_Prim_Topo_Type

Source: RenderCS

Size (in bits): 6

The following table defines the encoding of the Primitive Topology Type field. See 3D Pipeline for details, programming restrictions, diagrams and a discussion of the basic primitive types.

Value	Name	Description
00h	Reserved	
01h	3DPRIM_POINTLIST	
02h	3DPRIM_LINELIST	
03h	3DPRIM_LINESTRIP	
04h	3DPRIM_TRILIST	
05h	3DPRIM_TRISTRIP	
06h	3DPRIM_TRIFAN	
07h	3DPRIM_QUADLIST	The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.
08h	3DPRIM_QUADSTRIP	The QUADSTRIP topology is converted to POLYGON topology at the beginning of the 3D pipeline.
09h	3DPRIM_LINELIST_ADJ	
0Ah	3DPRIM_LINESTRIP_ADJ	
0Bh	3DPRIM_TRILIST_ADJ	
0Ch	3DPRIM_TRISTRIP_ADJ	
0Dh	3DPRIM_TRISTRIP_REVERSE	
0Eh	3DPRIM_POLYGON	
0Fh	3DPRIM_RECTLIST	
10h	3DPRIM_LINELOOP	The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline.
11h	3DPRIM_POINTLIST _BF	
12h	3DPRIM_LINESTRIP_CONT	
13h	3DPRIM_LINESTRIP_BF	
14h	3DPRIM_LINESTRIP_CONT_BF	
15h	Reserved	Reserved for HW use as TRISTRIP_ADJ_REV
16h	3DPRIM_TRIFAN_NOSTIPPLE	
17h	Reserved	Reserved for HW use as POLYGON_CONT
18h	Reserved	Reserved for HW use as LINESTRIP_ADJ_CONT
19h	Reserved	

3D_Prim_Topo_Type

1Ah-1Fh	Reserved	
20h	3DPRIM_PATCHLIST_1	List of 1-vertex patches
21h	3DPRIM_PATCHLIST_2	
22h	3DPRIM_PATCHLIST_3	
23h	3DPRIM_PATCHLIST_4	
24h	3DPRIM_PATCHLIST_5	
25h	3DPRIM_PATCHLIST_6	
26h	3DPRIM_PATCHLIST_7	
27h	3DPRIM_PATCHLIST_8	
28h	3DPRIM_PATCHLIST_9	
29h	3DPRIM_PATCHLIST_10	
2ah	3DPRIM_PATCHLIST_11	
2bh	3DPRIM_PATCHLIST_12	
2ch	3DPRIM_PATCHLIST_13	
2dh	3DPRIM_PATCHLIST_14	
2eh	3DPRIM_PATCHLIST_15	
2fh	3DPRIM_PATCHLIST_16	
30h	3DPRIM_PATCHLIST_17	
31h	3DPRIM_PATCHLIST_18	
32h	3DPRIM_PATCHLIST_19	
33h	3DPRIM_PATCHLIST_20	
34h	3DPRIM_PATCHLIST_21	
35h	3DPRIM_PATCHLIST_22	
36h	3DPRIM_PATCHLIST_23	
37h	3DPRIM_PATCHLIST_24	
38h	3DPRIM_PATCHLIST_25	
39h	3DPRIM_PATCHLIST_26	
3ah	3DPRIM_PATCHLIST_27	
3bh	3DPRIM_PATCHLIST_28	
3ch	3DPRIM_PATCHLIST_29	
3dh	3DPRIM_PATCHLIST_30	
3eh	3DPRIM_PATCHLIST_31	
3Fh	3DPRIM_PATCHLIST_32	List of 32-vertex patches

3D_Stencil_Operation

3D_Stencil_Operation	
Source:	RenderCS
Size (in bits):	3

Value	Name
0	STENCILOP_KEEP
1	STENCILOP_ZERO
2	STENCILOP_REPLACE
3	STENCILOP_INCRSAT
4	STENCILOP_DECRSAT
5	STENCILOP_INCR
6	STENCILOP_DECR
7	STENCILOP_INVERT

3D_Vertex_Component_Control

3D_Vertex_Component_Control		
Source:	RenderCS	
Size (in bits):	3	
Value	**Name**	**Description**
0	VFCOMP_NOSTORE	Don't store this component. (Not valid for Component 0, but can be used for Component 1-3). Once this setting is used for a component, all higher-numbered components (if any) MUST also use this setting. (I.e., no holes within any particular vertex element). VFCOMP_NOSTORE will not store a component if the SourceElementFormat is R64_PASSTHRU or R64G64_PASSTHRU and it is used on component 2 and 3 else 0 will be stored.
1	VFCOMP_STORE_SRC	Store corresponding component from format-converted source element. Storing a component that is not included in the Source Element Format results in an UNPREDICTABLE value being stored. VF will process Component Control fields within a VERTEX_ELEMENT_STATE structure sequentially, starting with Component 0 Control. For each Component Control field in this sequence, when VF detects (a) the Component Control field is set to STORE_SRC and (b) the component is not overwritten by an SGV, VF will store a component of the source vertex data into the destination component. The first such Component Control field satisfying this criteria will use Component 0 of the source vertex data, the second such Component Control field will use Component 1 of the source vertex data, and so on. Therefore, when a lower-numbered Component Control field (a) is set to something other than STORE_SRC (e.g., STORE_0) or (b) the component is overwritten with an SGV, the source vertex component used when a higher-numbered Component Control fields is set to STORE_SRC will be impacted.
2	VFCOMP_STORE_0	Store 0 (interpreted as 0.0f if accessed as a float value)
3	VFCOMP_STORE_1_FP	Store 1.0f
4	VFCOMP_STORE_1_INT	Store 0x1
5-6	-	Reserved
7	VFCOMP_STORE_PID	Store Primitive ID (as U32) Software can no longer use this encoding as PrimitiveID is passed down the FF pipeline - see explanation above.

AddrMode

AddrMode

Source: Eulsa

Size (in bits): 1

Addressing Mode

This field determines the addressing method of the operand. Normally the destination operand and each source operand each have a distinct addressing mode field.

When it is cleared, the register address of the operand is directly provided by bits in the instruction word. It is called a direct register addressing mode. When it is set, the register address of the operand is computed based on the address register value and an address immediate field in the instruction word. This is referred to as a register-indirect register addressing mode.

This field applies to the destination operand and the first source operand, src0. Support for src1 is device dependent. See Table XX (Indirect source addressing support available in device hardware) in ISA Execution Environment for details.

Programming Notes
Instructions with 3 source operands use Direct Addressing.

Value	Name	Description
0	Direct	'Direct' register addressing
1	Indirect	'Register-Indirect' (or in short 'Indirect'). Register-indirect register addressing

Attribute_Component_Format

Attribute_Component_Format		
Source:	RenderCS	
Size (in bits):	2	
Value	**Name**	**Description**
00b	disabled **[Default]**	All components disabled
01b	.xy	2D attribute, z and w components disabled
10b	.xyz	3D attribute, w components disabled
11b	.xyzw	4D attribute, no disabled components

ChanEn

ChanEn	
Source:	Eulsa
Size (in bits):	1

Description
Channel Enables Dst.ChanEn, used only for the destination operand and only in the Align16 Access Mode. Provides four channel enable bits applied modulo four to all ExecSize channels. For example, 0xF enables all channels, 0 disables all channels, 0xA enables odd-numbered channels, and so on. The assembler mnemonics are x, y, z, and w for channels 0, 1, 2, and 3 respectively. If MaskCtrl is 1 (mnemonic NoMask) then all channels are enabled regardless of the ChanEn value, equivalent to ChanEn of 0xF (xyzw). Predication and execution masking, in addition to ChanEn and MaskCtrl, determine what channels are actually written.

Value	Name
0	Write Disabled
1	Write Enabled **[Default]**

ChanSel

ChanSel

Source: Eulsa

Size (in bits): 2

Channel Select

This field controls the channel swizzle for a source operand. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3.2], [5.4] and [7,6] are for channel 0 (?x?), 1 (?y?), 2 (?z?), and 3 (?w?) in the group, respectively.

For example with an execution size of 8, r0.0<4>.zywz:f would assign the channels as follows: Chan0 = Data2, Chan1 = Data1, Chan2 = Data3, Chan3 = Data2; Chan4 = Data6, Chan5 = Data5, Chan6 = Data7, Chan7 = Data6.

This field only applies to source operand.

This field is only present in Align16 mode. It is not present for an immediate source operand.

The 2-bit Channel Selection field for each channel within the group of 4 is defined as the following.

Programming Notes
Note: When using channel select for 64-bit operands, the valid selects are .xy and .zw. This is required to pick a pair of DWords.

Value	Name	Description
00b	x	Channel 0 is selected for the corresponding execution channel
01b	y	Channel 1 is selected for the corresponding execution channel
10b	z	Channel 2 is selected for the corresponding execution channel
11b	w	Channel 3 is selected for the corresponding execution channel

COMPONENT_ENABLES

COMPONENT_ENABLES	
Source:	RenderCS
Size (in bits):	4
If enabled, the component will be stored in the URB.	

Value	Name
0000b	NONE
0001b	X
0010b	Y
0011b	XY
0100b	Z
0101b	XZ
0110b	YZ
0111b	XYZ
1000b	W
1001b	XW
1010b	YW
1011b	XYW
1100b	ZW
1101b	XZW
1110b	YZW
1111b	XYZW

CondModifier

CondModifier	
Source:	Eulsa
Size (in bits):	4

Conditional Modifier - This field sets the flag register based on the internal conditional signals output from the execution pipe such as sign, zero, overflow and NaNs, etc. If this field is set to 0000, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field may also be referred to as the flag destination control field. This field applies to all instructions except send, sendc, and math.

Value	Name	Description
0000b	None **[Default]**	Do Not modify Flag Register.
0001b	.z	Zero
0001b	.e	Equal
0010b	.nz	NotZero
0010b	.ne	NotEqual
0011b	.g	Greater-than
0100b	.ge	Greater-than-or-equal
0101b	.l	Less-than
0110b	.le	Less-than-or-equal
0111b	Reserved	
1000b	.o	Overflow
1001b	.u	Unordered with Computed NaN
1110b-1111b	Reserved	

DepCtrl

DepCtrl

Source: Eulsa

Size (in bits): 2

Destination Dependency Control

This field selectively disables destination dependency check and clear for this instruction. When it is set to 00, normal destination dependency control is performed for the instruction - hardware checks for destination hazards to ensure data integrity. Specifically, destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard will be cleared when the destination operands retire. When bit 10 is set (NoDDClr), the destination register scoreboard will NOT be cleared when the destination operands retire. When bit 11 is set (NoDDChk), hardware does not check for destination register dependency before the instruction is made ready for execution. NoDDClr and NoDDChk are not mutual exclusive. When this field is not all-zero, hardware does not protect against destination hazards for the instruction. This is typically used to assemble data in a fine grained fashion (e.g. matrix-vector compute with dot-product instructions), where the data integrity is guaranteed by software based on the intended usage of instruction sequences.

Value	Name	Description
00b	None **[Default]**	Destination dependency checked and cleared (normal)
01b	NoDDClr	Destination dependency checked but not cleared
10b	NoDDChk	Destination dependency not checked but cleared
11b	NoDDClr, NoDDChk	Destination dependency not checked and not cleared

DstType

DstType	
Source:	Eulsa
Size (in bits):	4

Destination Type
Numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send or sendc instruction, this field applies to CurrDst, the current destination operand. Three source instructions use a 3-bit encoding that allows fewer data types.

Value	Name	Description
0000b	:ud	Unsigned Doubleword (32-bit) Integer
0001b	:d	Signed Doubleword (32-bit) Integer
0010b	:uw	Unsigned Word (16-bit) Integer
0011b	:w	Signed Word (16-bit) Integer
0100b	:ub	Unsigned Byte (8-bit) Integer
0101b	:b	Signed Byte (8-bit) Integer
0110b	:df	Double-Precision (64-bit) Float
0111b	:f	Single-Precision (32-bit) Float
1000b	:uq	Unsigned Quadword (64-bit) Integer
1001b	:q	Signed Quadword (64-bit) Integer
1010b	:hf	Half (16-bit) Float
1011b-1111b	Reserved	

EU_OPCODE

EU_OPCODE	
Source:	Eulsa
Size (in bits):	7

Value	Name
40h	add
4Eh	addc
5h	and
0Ch	asr
42h	avg
18h	bfe
19h	bfi1
1Ah	bfi2
17h	bfrev
23h	brc
21h	brd
28h	break
2Ch	call
2Bh	calla
4Dh	cbit
10h	cmp
11h	cmpn
29h	cont
12h	csel
57h	dp2
56h	dp3
54h	dp4
55h	dph
24h	else
25h	endif
4Bh	fbh
4Ch	fbl
43h	frc
2Eh	goto
2Ah	halt

EU_OPCODE	
22h	if
0h	illegal
20h	jmpi
2Fh	join
59h	line
5Ch	lrp
4Ah	lzd
48h	mac
49h	mach
5Bh	mad
5Dh	madm
38h	math
1h	mov
3h	movi
41h	mul
7Eh	nop
4h	not
6h	or
5Ah	pln
2Dh	ret
45h	rndd
46h	rnde
44h	rndu
47h	rndz
50h	sad2
51h	sada2
2h	sel
31h	send
32h	sendc
33h	sends
34h	sendsc
9h	shl
8h	shr
0Ah	smov
4Fh	subb

EU_OPCODE	
30h	wait
27h	while
7h	xor

ExecSize

ExecSize	
Source:	Eulsa
Size (in bits):	3

Execution Size
This field determines the number of channels operating in parallel for this instruction.
The size cannot exceed the maximum number of channels allowed for the given data type.

Restriction
Restriction : An operand's Width must be less-than-or-equal to ExecSize

Value	Name	Programming Notes
000b	1 Channel (Scalar operation) **[Default]**	
001b	2 Channels	
010b	4 Channels	
011b	8 Channels	
100b	16 Channels	Restriction : 4-byte or smaller data types. Excludes DF, Q, and UQ types.
101b	32 Channels	Restriction : 2-byte or 1-byte data types. Excludes D, DF, F, Q, UD, and UQ types.
110b-111b	Reserved	

FC

FC	
Source:	Eulsa
Size (in bits):	4
Math Function Control	

Value	Name	Description
0000b	Reserved	
0001b	INV (reciprocal)	
0010b	LOG	
0011b	EXP	
0100b	SQRT	
0101b	RSQ	
0110b	SIN	
0111b	COS	
1000b	Reserved	
1001b	FDIV	
1010b	POW	
1011b	INT DIV BOTH	Return Quotient and Remainder
1100b	INT DIV QUOTIENT	Return Quotient Only
1101b	INT DIV REMAINDER	Return Remainder
1110b	INVM	
1111b	RSQRTM	

Fixed Function ID

FFID - Fixed Function ID

Source: BSpec

Size (in bits): 4

Fixed functions are hardware units that execute complex graphics or media command on behalf of application software. Some fixed functions send work down a pipeline through a series of fixed functions. Multiple fixed functions can be running at the same time. The GPU tracks activity from the fixed function with its FFID.

Programming Notes

Software does not specify the FFID, and does not normally use the FFID value. The FFID value is available to a EU thread in an ARF.

Value	Name	Description
00h	Null	
04h	Hull Shader	
05h	Domain Shader	
07h	General Purpose Thread Spawner	GPGPU command queue thread dispatcher
09h	Vertex Shader	
0Ah	Command Stream	
0Bh	Vertex Fetch	
0Ch	Geometry Shader	

HorzStride

HorzStride	
Source:	Eulsa
Size (in bits):	2

Description
Horizontal Stride This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand. This field applies to both destination and source operands. This field is not present for an immediate source operand.
A horizontal stride of 0 is used for a row that is one-element wide, useful when an instruction repeats a column value or repeats a scalar value. For example, adding a single column to every column in a 2D array or adding a scalar to every element in a 2D array uses HorzStride of 0. A horizontal stride of 1 indicates that elements are adjacent within a row. References to HorzStride in this volume normally reference the value not the encoding, so there are references to HorzStride of 4, which is encoded as 11b.

Value	Name
00b	0 elements
01b	1 elements
10b	2 elements
11b	4 elements

Performance Counter Report Formats

Performance Counter Report Formats

Source: BSpec

Size (in bits): 3

Value	Name
001b	
010b	
011b	
100b	
110b	
111b	

PredCtrl

PredCtrl	
Source:	Eulsa
Size (in bits):	4

Value	Name	Exists If
0000b	No Predication (normal) **[Default]**	
0001b	Sequential Flag Channel Mapping	
0010b	Replication swizzle .x	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0010b	.anyv (any from f0.0-f1.0 on the same channel)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0011b	Replication swizzle .y	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0011b	.allv (all of f0.0-f1.0 on the same channel)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0100b	Replication swizzle .z	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0100b	.any2h (any in group of 2 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0101b	Replication swizzle .w	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0101b	.all2h (all in group of 2 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0110b	.any4h	
0111b	.all4h	
1000b-1111b	Reserved	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
1000b	.any8h (any in group of 8 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1001b	.all8h (all in group of 8 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1010b	.any16h (any in group of 16 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1011b	.all16h (all in group of 16 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1100b	.any32h (any in group of 32 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1101b	.all32h (all in group of 32 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1110b-1111b	Reserved	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')

QtrCtrl

QtrCtrl

Source: EuIsa

Size (in bits): 2

Quarter Control
This field provides explicit control for ARF selection.
This field combined with ExecSize determines which channels are used for the ARF registers.
Along with NibCtrl, 1/8 DMask/VMask and ARF can be selected.

Programming Notes

NibCtrl is only allowed for SIMD4 instructions with a DF (Double Float) source or destination type.

Value	Name	Description	Programming Notes	Exists If
00b	1Q **[Default]**	Use first quarter for DMask/VMask. Use first half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
01b	2Q	Use second quarter for DMask/VMask. Use second half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
10b	3Q	Use third quarter for DMask/VMask. Use first half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
11b	4Q	Use fourth quarter for DMask/VMask. Use second half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
0	1H	Use first half for DMask/VMask. Use all channels for everything else.		([ExecSize]=='16') AND ([NibCtrl]=='0')
2	2H	Use second half for DMask/VMask. Use all channels for everything else.	Only allowed for SIMD16 instruction in Single Program Flow mode (SPF=1)	([ExecSize]=='16') AND ([NibCtrl]=='0')
0	1N	Use first 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
0	2N	Use second 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
1	3N	Use third 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
1	4N	Use fourth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
2	5N	Use fifth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')

QtrCtrl				
2	6N	Use sixth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
3	7N	Use seventh 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
3	8N	Use eighth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')

RegFile

RegFile

Source: Eulsa

Size (in bits): 2

Value	Name	Description	Programming Notes
00b	ARF	Architecture Register File	Restriction : Only allowed for src0 or destination
01b	GRF	General Register File - allowed for any source or destination	
10b	Reserved		
11b	IMM	Immediate operand	Restriction : Only allowed for the last source operand. Not allowed for the destination operand or for any other source operand. Note that for flow control instructions requiring two offsets, regfile of source0 is required to be immediate since the 64b for immediates occupy the DW2 and DW3

RepCtrl

RepCtrl	
Source:	Eulsa
Size (in bits):	1

Replicate Control. This field is only present in three-source instructions, for each of the three source operands. It controls replication of the starting channel to all channels in the execution size. This is applicable to 32b datatypes. 16b and 64b datatypes cannot use the replicate control.

Value	Name
0	No replication
1	Replicate across all channels

SFID

SFID

Source: Eulsa

Size (in bits): 4

The following table lists the assignments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a 'send' instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function.

Note: The initial intention was to combine these two ID namespaces, so that (theoretically) an agent (such as the Thread Spawner) that served both as a Shared Function and Fixed Function would have a single, unique 4-bit ID encoding. However, this combination is not a requirement of the architecture.

Programming Notes
SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.
SFID_DP_DC1, SFID_DP_DC2, SFID_DP_DC3 are extensions of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.

Value	Name	Description	Programming Notes
0000b	SFID_NULL	Null	
0001b	Reserved	Reserved	
0010b	SFID_SAMPLER	Sampler	
0011b	SFID_GATEWAY	Message Gateway	
0100b	SFID_DP_DC2	Data Cache Data Port 2	
0101b	SFID_DP_RC	Render Cache Data Port	
0110b	SFID_URB	URB	
0111b	SFID_SPAWNER	Thread Spawner	
1000b	SFID_VME	Video Motion Estimation	
1001b	SFID_DP_DCRO	Data Cache Read Only Data Port	
1010b	SFID_DP_DC0	Data Cache Data Port	
1011b	SFID_PI	Pixel Interpolator	
1100b	SFID_DP_DC1	Data Cache Data Port 1	SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.
1101b	SFID_CRE	Check and Refinement Engine	

SFID			
1110b-1111b	Reserved		

Shader Channel Select

Shader Channel Select

Source: BSpec

Size (in bits): 3

Value	Name	Description
0	ZERO	
1	ONE	
2	Reserved	
3	Reserved	
4	RED	Shader channel is set to surface red channel
5	GREEN	Shader channel is set to surface green channel
6	BLUE	Shader channel is set to surface blue channel
7	ALPHA	Shader channel is set to surface alpha channel

Shared Function ID

SFID - Shared Function ID

Source: BSpec

Size (in bits): 4

The Shared Function ID is specified in the extended descriptor of the SEND instruction. The use of an unknown SFID is illegal programming and will have undefined results.

Value	Name
00h	Null
02h	Sampler
03h	Message Gateway
4h	Data Cache Data Port 2
05h	Render Cache Data Port
06h	URB
07h	Thread Spawner
08h	Video Motion Estimation
09h	Data Cache Read Only Data Port
0Ah	Data Cache Data Port 0
0Bh	Pixel Interpolator
0Ch	Data Cache Data Port 1
0Dh	Check and Refinement Engine

SIMD Mode

SIMD Mode

Source: BSpec

Size (in bits): 2

Value	Name
0	SIMD8D / SIMD4x2
1	SIMD8
2	SIMD16
3	SIMD32/64

SrcImmType

SrcImmType

Source: EuIsa

Size (in bits): 4

Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands.

The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator.

Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF.

Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions.

In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW.

Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding.

The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands.

Value	Name	Description
0000b	:ud	Unsigned Doubleword Integer
0001b	:d	Signed Doubleword Integer
0010b	:uw	Unsigned Word Integer
0011b	:w	Signed Word Integer
0100b	:uv	Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer.
0101b	:vf	Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number
0110b	:v	Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer
0111b	:f	Single-Precision (32-bit) Float
1000b	:uq	Unsigned Quadword Integer
1001b	:q	Signed Quadword Integer
1010b	:df	Double-Precision Float (64-bit)
1011b	:hf	Half-Precision Float (16-bit)
1100b-1111b	Reserved	

SrcIndex

SrcIndex		
Source:	Eulsa	
Size (in bits):	5	
Value	**Name**	**Description**
0	000000000000	dir | <0;1,0>
1	000000000010	(-) | dir | <0;1,0>
2	000000010000	dir | <0;>.zx
3	000000010010	(-) | dir | <0;>.zx
4	000000011000	dir | <0;>.wx
5	000000100000	dir | <0;>.xy
6	000000101000	dir | <0;>.yy
7	000001001000	dir | <0;4,1>
8	000001010000	dir | <0;>.zz
9	000001110000	dir | <0;>.zw
10	000001111000	dir | <0;8,4> / dir | <0;>.ww
11	001100000000	dir | <4;>.xx
12	001100000010	(-) | dir | <4;>.xx
13	001100001000	dir | <4;>.yx
14	001100010000	dir | <4;>.zx
15	001100010010	(-) | dir | <4;>.zx
16	001100100000	dir | <4;>.xy
17	001100101000	dir | <4;>.yy
18	001100111000	dir | <4;>.wy
19	001101000000	dir | <4;4,0>
20	001101000010	(-) | dir | <4;4,0>
21	001101001000	dir | <4;>.yz
22	001101010000	dir | <4;>.zz
23	001101100000	dir | <4;>.xw
24	001101101000	dir | <4;>.yw
25	001101110000	dir | <4;>.zw
26	001101110001	(abs) | dir | <4;>.zw
27	001101111000	dir | <4;>.ww
28	010001101000	dir | <8;8,1>
29	010001101001	(abs) | dir | <8;8,1>

SrcIndex		
30	010001101010	(-) | dir | <8;8,1>
31	010110001000	dir | <16;16,1>

SrcMod

SrcMod

Source: Eulsa

Size (in bits): 2

Description
Source Modifier This field specifies the numeric modification of a source operand. The value of each data element of a source operand can optionally have its absolute value taken and/or its sign inverted prior to delivery to the execution pipe. The absolute value is prior to negate such that a guaranteed negative value can be produced. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.
When used with logic instructions (and, not, or, xor), this field indicates whether the source bits are inverted (bitwise NOT) before delivery to the execution pipe, regardless of the source type.

Value	Name	Description
00b	No modification	
01b	abs	Absolute value Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)
10b	negate	Negate Logic instructions: Bitwise NOT, inverting the source bits
11b	negate of abs	Negate of the absolute (forced negative value) Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)

SrcType

SrcType

Source: EuIsa

Size (in bits): 4

Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands.

The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator.

Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF.

Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions.

In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW.

Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding.

The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands.

Value	Name	Description
0000b	:ud	Unsigned Doubleword (32-bit) Integer
0001b	:d	Signed Doubleword (32-bit) Integer
0010b	:uw	Unsigned Word (16-bit) Integer
0011b	:w	Signed Word (16-bit) Integer
0100b	:ub	Unsigned Byte (8-bit) Integer
0101b	:b	Signed Byte (8-bit) Integer
0110b	:df	Double-Precision (64-bit) Float
0111b	:f	Single-Precision (32-bit) Float
1000b	:uq	Unsigned Quadword (64-bit) Integer
1001b	:q	Signed Quadword (64-bit) Integer
1010b	:hf	Half (16-bit) Float
1011b-1111b	Reserved	

SURFACE_FORMAT

SURFACE_FORMAT

Source: BSpec

Size (in bits): 9

The following table indicates the supported surface formats and the 9-bit encoding for each. Note that some of these formats are used not only by the Sampling Engine, but also by the Data Port and the Vertex Fetch unit.

Value	Name	Bits Per Element (BPE)	Description
000h	R32G32B32A32_FLOAT	128	
001h	R32G32B32A32_SINT	128	
002h	R32G32B32A32_UINT	128	
003h	R32G32B32A32_UNORM	128	
004h	R32G32B32A32_SNORM	128	
005h	R64G64_FLOAT	128	
006h	R32G32B32X32_FLOAT	128	
007h	R32G32B32A32_SSCALED	128	
008h	R32G32B32A32_USCALED	128	
020h	R32G32B32A32_SFIXED	128	
021h	R64G64_PASSTHRU	128	
040h	R32G32B32_FLOAT	96	
041h	R32G32B32_SINT	96	
042h	R32G32B32_UINT	96	
043h	R32G32B32_UNORM	96	
044h	R32G32B32_SNORM	96	
045h	R32G32B32_SSCALED	96	
046h	R32G32B32_USCALED	96	
050h	R32G32B32_SFIXED	96	
080h	R16G16B16A16_UNORM	64	
081h	R16G16B16A16_SNORM	64	
082h	R16G16B16A16_SINT	64	
083h	R16G16B16A16_UINT	64	
084h	R16G16B16A16_FLOAT	64	
085h	R32G32_FLOAT	64	
086h	R32G32_SINT	64	
087h	R32G32_UINT	64	

SURFACE_FORMAT			
088h	R32_FLOAT_X8X24_TYPELESS	64	
089h	X32_TYPELESS_G8X24_UINT	64	
08Ah	L32A32_FLOAT	64	
08Bh	R32G32_UNORM	64	
08Ch	R32G32_SNORM	64	
08Dh	R64_FLOAT	64	
08Eh	R16G16B16X16_UNORM	64	
08Fh	R16G16B16X16_FLOAT	64	
090h	A32X32_FLOAT	64	
091h	L32X32_FLOAT	64	
092h	I32X32_FLOAT	64	
093h	R16G16B16A16_SSCALED	64	
094h	R16G16B16A16_USCALED	64	
095h	R32G32_SSCALED	64	
096h	R32G32_USCALED	64	
0A0h	R32G32_SFIXED	64	
0A1h	R64_PASSTHRU	64	
0C0h	B8G8R8A8_UNORM	32	
0C1h	B8G8R8A8_UNORM_SRGB	32	
0C2h	R10G10B10A2_UNORM	32	
0C3h	R10G10B10A2_UNORM_SRGB	32	
0C4h	R10G10B10A2_UINT	32	
0C5h	R10G10B10_SNORM_A2_UNORM	32	
0C7h	R8G8B8A8_UNORM	32	
0C8h	R8G8B8A8_UNORM_SRGB	32	
0C9h	R8G8B8A8_SNORM	32	
0CAh	R8G8B8A8_SINT	32	
0CBh	R8G8B8A8_UINT	32	
0CCh	R16G16_UNORM	32	
0CDh	R16G16_SNORM	32	
0CEh	R16G16_SINT	32	
0CFh	R16G16_UINT	32	
0D0h	R16G16_FLOAT	32	
0D1h	B10G10R10A2_UNORM	32	
0D2h	B10G10R10A2_UNORM_SRGB	32	

SURFACE_FORMAT

0D3h	R11G11B10_FLOAT	32	
0D6h	R32_SINT	32	
0D7h	R32_UINT	32	
0D8h	R32_FLOAT	32	
0D9h	R24_UNORM_X8_TYPELESS	32	
0DAh	X24_TYPELESS_G8_UINT	32	
0DDh	L32_UNORM	32	
0DEh	A32_UNORM	32	
0DFh	L16A16_UNORM	32	
0E0h	I24X8_UNORM	32	
0E1h	L24X8_UNORM	32	
0E2h	A24X8_UNORM	32	
0E3h	I32_FLOAT	32	
0E4h	L32_FLOAT	32	
0E5h	A32_FLOAT	32	
0E6h	X8B8_UNORM_G8R8_SNORM	32	
0E7h	A8X8_UNORM_G8R8_SNORM	32	
0E8h	B8X8_UNORM_G8R8_SNORM	32	
0E9h	B8G8R8X8_UNORM	32	
0EAh	B8G8R8X8_UNORM_SRGB	32	
0EBh	R8G8B8X8_UNORM	32	
0ECh	R8G8B8X8_UNORM_SRGB	32	
0EDh	R9G9B9E5_SHAREDEXP	32	
0EEh	B10G10R10X2_UNORM	32	
0F0h	L16A16_FLOAT	32	
0F1h	R32_UNORM	32	
0F2h	R32_SNORM	32	
0F3h	R10G10B10X2_USCALED	32	
0F4h	R8G8B8A8_SSCALED	32	
0F5h	R8G8B8A8_USCALED	32	
0F6h	R16G16_SSCALED	32	
0F7h	R16G16_USCALED	32	
0F8h	R32_SSCALED	32	
0F9h	R32_USCALED	32	
100h	B5G6R5_UNORM	16	

SURFACE_FORMAT			
101h	B5G6R5_UNORM_SRGB	16	
102h	B5G5R5A1_UNORM	16	
103h	B5G5R5A1_UNORM_SRGB	16	
104h	B4G4R4A4_UNORM	16	
105h	B4G4R4A4_UNORM_SRGB	16	
106h	R8G8_UNORM	16	
107h	R8G8_SNORM	16	
108h	R8G8_SINT	16	
109h	R8G8_UINT	16	
10Ah	R16_UNORM	16	
10Bh	R16_SNORM	16	
10Ch	R16_SINT	16	
10Dh	R16_UINT	16	
10Eh	R16_FLOAT	16	
10Fh	A8P8_UNORM_PALETTE0	16	
110h	A8P8_UNORM_PALETTE1	16	
111h	I16_UNORM	16	
112h	L16_UNORM	16	
113h	A16_UNORM	16	
114h	L8A8_UNORM	16	
115h	I16_FLOAT	16	
116h	L16_FLOAT	16	
117h	A16_FLOAT	16	
118h	L8A8_UNORM_SRGB	16	
119h	R5G5_SNORM_B6_UNORM	16	
11Ah	B5G5R5X1_UNORM	16	
11Bh	B5G5R5X1_UNORM_SRGB	16	
11Ch	R8G8_SSCALED	16	
11Dh	R8G8_USCALED	16	
11Eh	R16_SSCALED	16	
11Fh	R16_USCALED	16	
122h	P8A8_UNORM_PALETTE0	16	
123h	P8A8_UNORM_PALETTE1	16	
124h	A1B5G5R5_UNORM	16	
125h	A4B4G4R4_UNORM	16	

SURFACE_FORMAT

126h	L8A8_UINT	16	
127h	L8A8_SINT	16	
140h	R8_UNORM	8	
141h	R8_SNORM	8	
142h	R8_SINT	8	
143h	R8_UINT	8	
144h	A8_UNORM	8	
145h	I8_UNORM	8	
146h	L8_UNORM	8	
147h	P4A4_UNORM_PALETTE0	8	
148h	A4P4_UNORM_PALETTE0	8	
149h	R8_SSCALED	8	
14Ah	R8_USCALED	8	
14Bh	P8_UNORM_PALETTE0	8	
14Ch	L8_UNORM_SRGB	8	
14Dh	P8_UNORM_PALETTE1	8	
14Eh	P4A4_UNORM_PALETTE1	8	
14Fh	A4P4_UNORM_PALETTE1	8	
150h	Y8_UNORM	8	
152h	L8_UINT	8	
153h	L8_SINT	8	
154h	I8_UINT	8	
155h	I8_SINT	8	
180h	DXT1_RGB_SRGB	0	
181h	R1_UNORM	1	
182h	YCRCB_NORMAL	0	
183h	YCRCB_SWAPUVY	0	
184h	P2_UNORM_PALETTE0	2	
185h	P2_UNORM_PALETTE1	2	
186h	BC1_UNORM	0	(DXT1)
187h	BC2_UNORM	0	(DXT2/3)
188h	BC3_UNORM	0	(DXT4/5)
189h	BC4_UNORM	0	
18Ah	BC5_UNORM	0	
18Bh	BC1_UNORM_SRGB	0	(DXT1_SRGB)

SURFACE_FORMAT			
18Ch	BC2_UNORM_SRGB	0	(DXT2/3_SRGB)
18Dh	BC3_UNORM_SRGB	0	(DXT4/5_SRGB)
18Eh	MONO8	1	
18Fh	YCRCB_SWAPUV	0	
190h	YCRCB_SWAPY	0	
191h	DXT1_RGB	0	
192h	FXT1	0	
193h	R8G8B8_UNORM	24	
194h	R8G8B8_SNORM	24	
195h	R8G8B8_SSCALED	24	
196h	R8G8B8_USCALED	24	
197h	R64G64B64A64_FLOAT	256	
198h	R64G64B64_FLOAT	192	
199h	BC4_SNORM	0	
19Ah	BC5_SNORM	0	
19Bh	R16G16B16_FLOAT	48	
19Ch	R16G16B16_UNORM	48	
19Dh	R16G16B16_SNORM	48	
19Eh	R16G16B16_SSCALED	48	
19Fh	R16G16B16_USCALED	48	
1A1h	BC6H_SF16	0	
1A2h	BC7_UNORM	0	
1A3h	BC7_UNORM_SRGB	0	
1A4h	BC6H_UF16	0	
1A5h	PLANAR_420_8	0	
1A8h	R8G8B8_UNORM_SRGB	24	
1A9h	ETC1_RGB8	0	
1AAh	ETC2_RGB8	0	
1ABh	EAC_R11	0	
1ACh	EAC_RG11	0	
1ADh	EAC_SIGNED_R11	0	
1AEh	EAC_SIGNED_RG11	0	
1AFh	ETC2_SRGB8	0	
1B0h	R16G16B16_UINT	48	
1B1h	R16G16B16_SINT	48	

SURFACE_FORMAT

1B2h	R32_SFIXED	32	
1B3h	R10G10B10A2_SNORM	32	
1B4h	R10G10B10A2_USCALED	32	
1B5h	R10G10B10A2_SSCALED	32	
1B6h	R10G10B10A2_SINT	32	
1B7h	B10G10R10A2_SNORM	32	
1B8h	B10G10R10A2_USCALED	32	
1B9h	B10G10R10A2_SSCALED	32	
1BAh	B10G10R10A2_UINT	32	
1BBh	B10G10R10A2_SINT	32	
1BCh	R64G64B64A64_PASSTHRU	256	
1BDh	R64G64B64_PASSTHRU	192	
1C0h	ETC2_RGB8_PTA	0	
1C1h	ETC2_SRGB8_PTA	0	
1C2h	ETC2_EAC_RGBA8	0	
1C3h	ETC2_EAC_SRGB8_A8	0	
1C8h	R8G8B8_UINT	24	
1C9h	R8G8B8_SINT	24	
1FFh	RAW	0	

Texture Coordinate Mode

Texture Coordinate Mode

Source: BSpec

Size (in bits): 3

Value	Name	Description
0h	WRAP	Map is repeated in the U direction
1h	MIRROR	Map is mirrored in the U direction
2h	CLAMP	Map is clamped to the edges of the accessed map
3h	CUBE	For cube-mapping, filtering in edges access adjacent map faces
4h	CLAMP_BORDER	Map is infinitely extended with the border color
5h	MIRROR_ONCE	Map is mirrored once about origin, then clamped
6h	HALF_BORDER	Map is infinitely extended with the average of the nearest edge texel and the border color
7h	Reserved	

ThreadCtrl

ThreadCtrl

Source: Eulsa

Size (in bits): 2

Thread Control
This field provides explicit control for thread switching.

Value	Name	Description
00b	Normal	Up to the GEN execution units to manage thread switching. This is the normal (and unnamed) mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. In another example, if the current instruction is ready to go, however, there is another thread with higher priority that also has an instruction ready, the EU switches to that thread. Execution may or may not be preempted by another thread following this instruction.
01b	Atomic	Prevent any thread switch immediately following this instruction. Always execute the next instruction (which may not be next sequentially if the current instruction branches). The next instruction gets highest priority in the thread arbitration for the execution pipelines.
10b	Switch	A forced thread switch occurs after the current instruction is executed and before the next instruction. In addition, a long delay (longer than the execution pipe latency) is introduced for the current thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. Switch is designed primarily as a safety feature in case there are race conditions for certain instructions. Force a switch to another thread after this instruction and before the next instruction.
11b	Reserved	

VertStride

VertStride	
Source:	Eulsa
Size (in bits):	4

Vertical Stride
The field provides the vertical stride of the register region in unit of data elements for an operand.
Encoding of this field provides values of 0 or powers of 2, ranging from 1 to 32 elements. Larger values are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total).
Special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (AddrMode = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute the addresses. Each address sub-register provides the origin for a row of data element. The number of address sub-registers used is determined by the division of ExecSize of the instruction by the Width fields of the operand.
This field only applies to source operand. It does not apply to destination.
This field is not present for an immediate source operand.

Programming Notes
Note 1: Vertical Stride larger than 32 is not allowed due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total).
Note 2: In Align16 access mode, as encoding 0xF is reserved, only single-index indirect addressing is supported.
Note 3: If indirect address is supported for src1, encoding 0xF is reserved for src1 - only single-index indirect addressing is supported.
Note 4: Encoding 0010 applies for QWord-size operands.

Value	Name	Programming Notes
0000b	0 elements	
0001b	1 element	Restriction : Align1 mode only.
0010b	2 elements	
0011b	4 elements	
0100b	8 elements	Restriction : Align1 mode only.
0101b	16 elements	Restriction : Applies to byte or word operand only. Align1 mode only.
0110b	32 elements	Restriction : Applies to byte operand only. Align1 mode only.
0111b-1110b	Reserved	
1111b	VxH or Vx1 mode	Restriction : Only valid for register-indirect addressing in Align1 mode.

Width

Width

Source: Eulsa

Size (in bits): 3

This field specifies the number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the ExecSize field of the instruction.
This field only applies to source operand. It does not apply to destination.
This field is not present for an immediate source operand.

Programming Notes
Note that with ExecSize of 32, because the maximum Width is 16, there are at least two rows in a source region.

Value	Name
000b	1 elements
001b	2 elements
010b	4 elements
011b	8 elements
100b	16 elements
101b-111b	Reserved

WRAP_SHORTEST_ENABLE

WRAP_SHORTEST_ENABLE	
Source:	RenderCS
Size (in bits):	4

This state selects which components (if any) of Attribute [n] are to be interpolated in a "wrap shortest" fashion. Operation is UNDEFINED if any of these bits are set and the Constant Interpolation Enable bit associated with this attribute is set. Note that wrap-shortest interpolation is only supported for Attributes 0-15.

Value	Name	Description
0001b	X	Wrap Shortest X Component
0010b	Y	Wrap Shortest Y Component
0011b	XY	Wrap Shortest XY Components
0100b	Z	Wrap Shortest Z Component
0101b	XZ	Wrap Shortest XZ Components
0110b	YZ	Wrap Shortest YZ Components
0111b	XYZ	Wrap Shortest XYZ Components
1000b	W	Wrap Shortest W Component
1001b	XW	Wrap Shortest XW Components
1010b	YW	Wrap Shortest YW Components
1011b	XYW	Wrap Shortest XYW Components
1100b	ZW	Wrap Shortest ZW Components
1101b	XZW	Wrap Shortest XZW Components
1110b	YZW	Wrap Shortest YZW Components
1111b	XYZW	Wrap Shortest XYZW Components

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 2c: Command Reference: Registers
Part 1 – Registers A through L

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Active Head Pointer Register....1
Advanced Scheduler Reset Request Messages....3
Aggregate_Perf_Counter_A31....5
Aggregate_Perf_Counter_A32....6
Aggregate_Perf_Counter_A33....7
Aggregate_Perf_Counter_A34....8
Aggregate_Perf_Counter_A35....9
Aggregate Perf Counter A0....10
Aggregate Perf Counter A0 Upper DWord....11
Aggregate Perf Counter A1....12
Aggregate Perf Counter A1 Upper DWord....13
Aggregate Perf Counter A2....14
Aggregate Perf Counter A2 Upper DWord....15
Aggregate Perf Counter A3....16
Aggregate Perf Counter A3 Upper DWord....17
Aggregate Perf Counter A4....18
Aggregate Perf Counter A4 Lower DWord Free....19
Aggregate Perf Counter A4 Upper DWord....20
Aggregate Perf Counter A4 Upper DWord Free....21
Aggregate Perf Counter A5....22
Aggregate Perf Counter A5 Upper DWord....23
Aggregate Perf Counter A6....24
Aggregate Perf Counter A6 Lower DWord Free....25
Aggregate Perf Counter A6 Upper DWord....26
Aggregate Perf Counter A6 Upper DWord Free....27
Aggregate Perf Counter A7....28
Aggregate Perf Counter A7 Upper DWord....29
Aggregate Perf Counter A8....30
Aggregate Perf Counter A8 Upper DWord....31
Aggregate Perf Counter A9....32
Aggregate Perf Counter A9 Upper DWord....33

Aggregate Perf Counter A10 34
Aggregate Perf Counter A10 Upper DWord 35
Aggregate Perf Counter A11 36
Aggregate Perf Counter A11 Upper DWord 37
Aggregate Perf Counter A12 38
Aggregate Perf Counter A12 Upper DWord 39
Aggregate Perf Counter A13 40
Aggregate Perf Counter A13 Upper DWord 41
Aggregate Perf Counter A14 42
Aggregate Perf Counter A14 Upper DWord 43
Aggregate Perf Counter A15 44
Aggregate Perf Counter A15 Upper DWord 45
Aggregate Perf Counter A16 46
Aggregate Perf Counter A16 Upper DWord 47
Aggregate Perf Counter A17 48
Aggregate Perf Counter A17 Upper DWord 49
Aggregate Perf Counter A18 50
Aggregate Perf Counter A18 Upper DWord 51
Aggregate Perf Counter A19 52
Aggregate Perf Counter A19 Lower DWord Free 53
Aggregate Perf Counter A19 Upper DWord 54
Aggregate Perf Counter A19 Upper DWord Free 55
Aggregate Perf Counter A20 56
Aggregate Perf Counter A20 Lower DWord Free 57
Aggregate Perf Counter A20 Upper DWord 58
Aggregate Perf Counter A20 Upper DWord Free 59
Aggregate Perf Counter A21 60
Aggregate Perf Counter A21 Upper DWord 61
Aggregate Perf Counter A22 62
Aggregate Perf Counter A22 Upper DWord 63
Aggregate Perf Counter A23 64
Aggregate Perf Counter A23 Upper DWord 65
Aggregate Perf Counter A24 66

Aggregate Perf Counter A24 Upper DWord....67
Aggregate Perf Counter A25....68
Aggregate Perf Counter A25 Upper DWord....69
Aggregate Perf Counter A26....70
Aggregate Perf Counter A26 Upper DWord....71
Aggregate Perf Counter A27....72
Aggregate Perf Counter A27 Upper DWord....73
Aggregate Perf Counter A28....74
Aggregate Perf Counter A28 Upper DWord....75
Aggregate Perf Counter A29....76
Aggregate Perf Counter A29 Upper DWord....77
Aggregate Perf Counter A30....78
Aggregate Perf Counter A30 Upper DWord....79
Aggregate Perf Counter A31 Upper DWord....80
All Engine Fault Register....81
ARB_CTL....82
ARB_CTL2....84
Arbiter Control Register....87
Arbiter Mode Control Register....90
ASL Storage....93
ATS Capability....94
ATS Control....95
ATS Extended Capability Header....96
AUD_CONFIG....97
AUD_DIP_ELD_CTRL_ST....99
AUD_EDID_DATA....102
AUD_FREQ_CNTRL....104
AUD_INFOFR....106
AUD_M_CTS_ENABLE....107
AUD_MISC_CTRL....109
AUD_PIN_ELD_CP_VLD....111
AUD_PIN_PIPE_CONN_ENTRY_LNGTH....114
AUD_PIPE_CONN_SEL_CTRL....115

AUD_PWRST 116
AUD_RID 119
AUD_VID_DID 120
AUD_WNIC_PCR_CNTRL 121
Audio Codec Interrupt Definition 122
Auto Draw End Offset 124
AVC GAM Slave Counter High part 125
AVC GAM Slave Counter Low part 126
Base Data of Stolen Memory 127
Base of GTT Stolen Memory 128
Batch Address Difference Register 129
Batch Buffer Head Pointer Preemption Register 130
Batch Buffer Head Pointer Register 132
Batch Buffer Per Context Pointer 134
Batch Buffer Start Head Pointer Register 138
Batch Buffer Start Upper Head Pointer Register 139
Batch Buffer State Register 140
Batch Buffer Upper Head Pointer Preemption Register 142
Batch Buffer Upper Head Pointer Register 143
Batch Offset Register 144
BCS Context Sizes 146
BCS Ring Buffer Next Context ID Register 147
BCS SW Control 148
BITPLANE CYCLE CONTROL REGISTER 150
Bitstream Output Bit Count for the last Syntax Element Report Register 151
Bitstream Output Byte Count Per Slice Report Register 152
Bitstream Output Minimal Size Padding Count Report Register 153
BLC_PWM_CTL 154
BLC_PWM_DATA 156
Blitter MOCS Register0 157
Blitter MOCS Register1 159
Blitter MOCS Register2 161
Blitter MOCS Register3 163

Blitter MOCS Register4 165
Blitter MOCS Register5 167
Blitter MOCS Register6 169
Blitter MOCS Register7 171
Blitter MOCS Register8 173
Blitter MOCS Register9 175
Blitter MOCS Register10 177
Blitter MOCS Register11 179
Blitter MOCS Register12 181
Blitter MOCS Register13 183
Blitter MOCS Register14 185
Blitter MOCS Register15 187
Blitter MOCS Register16 189
Blitter MOCS Register17 191
Blitter MOCS Register18 193
Blitter MOCS Register19 195
Blitter MOCS Register20 197
Blitter MOCS Register21 199
Blitter MOCS Register22 201
Blitter MOCS Register23 203
Blitter MOCS Register24 205
Blitter MOCS Register25 207
Blitter MOCS Register26 209
Blitter MOCS Register27 211
Blitter MOCS Register28 213
Blitter MOCS Register29 215
Blitter MOCS Register30 217
Blitter MOCS Register31 219
Blitter MOCS Register32 221
Blitter MOCS Register33 223
Blitter MOCS Register34 225
Blitter MOCS Register35 227
Blitter MOCS Register36 229

Blitter MOCS Register37 231
Blitter MOCS Register38 233
Blitter MOCS Register39 235
Blitter MOCS Register40 237
Blitter MOCS Register41 239
Blitter MOCS Register42 241
Blitter MOCS Register43 243
Blitter MOCS Register44 245
Blitter MOCS Register45 247
Blitter MOCS Register46 249
Blitter MOCS Register47 251
Blitter MOCS Register48 253
Blitter MOCS Register49 255
Blitter MOCS Register50 257
Blitter MOCS Register51 259
Blitter MOCS Register52 261
Blitter MOCS Register53 263
Blitter MOCS Register54 265
Blitter MOCS Register55 267
Blitter MOCS Register56 269
Blitter MOCS Register57 271
Blitter MOCS Register58 273
Blitter MOCS Register59 275
Blitter MOCS Register60 277
Blitter MOCS Register61 279
Blitter MOCS Register62 281
Blitter MOCS Register63 283
Blitter TLB Control Register 285
BLT Context Element Descriptor (High Part) 286
BLT Context Element Descriptor (Low Part) 287
BLT Fault Counter 288
BLT Fixed Counter 289
BLT PDP0/PML4/PASID Descriptor (High Part) 290

BLT PDP0/PML4/PASID Descriptor (Low Part) 291
BLT PDP1 Descriptor Register (High Part) 292
BLT PDP1 Descriptor Register (Low Part) 293
BLT PDP2 Descriptor Register (High Part) 294
BLT PDP2 Descriptor Register (Low Part) 295
BLT PDP3 Descriptor Register (High Part) 296
BLT PDP3 Descriptor Register (Low Part) 297
Boolean_Counter_B0 298
Boolean_Counter_B1 299
Boolean_Counter_B2 300
Boolean_Counter_B3 301
Boolean_Counter_B4 302
Boolean_Counter_B5 303
Boolean_Counter_B6 304
Boolean_Counter_B7 305
BOOT VECTOR 306
BTB Not Consumed By RCS 307
BTP Commands Parsed By RCS 308
Built In Self Test 309
Cache Line Size 310
Cache Mode Register 0 311
Cache Mode Register 1 314
Capabilities A 319
Capabilities B 322
Capabilities Control 325
Capabilities Pointer 326
Capability Identifier 327
CDCLK_CTL 328
CGE_CTRL 330
CGE_WEIGHT 331
Class Code 334
Clipper Invocation Counter 335
Clipper Primitives Counter 336

Clock Gating Messages 337
Color/Depth Write FIFO Watermarks 339
Config to MCI HI 340
Config to MCI LO 341
Config to MCI STATUS1 342
Configuration Register0 for RPMunit 343
Configuration Register1 for RPMunit 344
Configuration Register for RCPunit 345
Context Load Protocol Register BLT 347
Context Load Protocol Register CS 350
Context Load Protocol Register VCS0 353
Context Load Protocol Register VCS1 356
Context Load Protocol Register VEBX 359
Context Restore Request To TDL 362
Context Save Request To TDL 363
Context Sizes 364
Context Status1 for RCS-BE 365
Context Status Buffer Contents 367
Context Timestamp Count 369
Control Register for Fault and Halt 370
Count Active Channels Dispatched 372
CSC_COEFF 373
CSC_MODE 375
CSC_POSTOFF 377
CSC_PREOFF 379
CSPREEMPT 381
CTX REG 1 382
CTX reg 2 383
CUR_BASE 384
CUR_CTL 386
CUR_FBC_CTL 390
CUR_PAL 392
CUR_POS 394

CUR_SURFLIVE 396
Current Context Register 397
Customizable Event Creation 0-0 400
Customizable Event Creation 1-0 402
Customizable Event Creation 1-1 404
Customizable Event Creation 2-0 405
Customizable Event Creation 2-1 407
Customizable Event Creation 3-0 408
Customizable Event Creation 3-1 410
Customizable Event Creation 4-0 411
Customizable Event Creation 5-0 413
Customizable Event Creation 5-1 415
Customizable Event Creation 6-0 416
Customizable Event Creation 6-1 418
Customizable Event Creation 7-0 419
Customizable Event Creation 7-1 421
CVS TLB LRA 0 422
CVS TLB LRA 1 424
CVS TLB LRA 2 426
DATAM 427
DATAN 429
DBUF_CTL 430
DBUF_ECC_STAT 432
DC_STATE_EN 434
DDI_AUX_CTL 436
DDI_AUX_DATA 439
DDI_AUX_MUTEX 440
DDI_BUF_CTL 442
DDI_BUF_TRANS 446
DE_PIPE_INTERRUPT 448
DE_RR_DEST 451
DE_RRMR 452
Decouple Register 0 DW0 456

Decouple Register 0 DW1 457
Decouple Register 1 DW0 458
Decouple Register 1 DW1 459
Decouple Register 2 DW0 460
Decouple Register 2 DW1 461
Decouple Register 3 DW0 462
Decouple Register 3 DW1 463
Decouple Register 4 DW0 464
Decouple Register 4 DW1 465
Decouple Register 5 DW0 466
Decouple Register 5 DW1 467
Decouple Register 6 DW0 468
Decouple Register 6 DW1 469
Decouple Register 7 DW0 470
Decouple Register 7 DW1 471
Decouple Register 8 DW0 472
Decouple Register 8 DW1 473
Decouple Register 9 DW0 474
Decouple Register 9 DW1 475
Decouple Register 10 DW0 476
Decouple Register 10 DW1 477
Decouple Register 11 DW0 478
Decouple Register 11 DW1 479
Decouple Register 12 DW0 480
Decouple Register 12 DW1 481
Decouple Register 13 DW0 482
Decouple Register 13 DW1 483
Decouple Register 14 DW0 484
Decouple Register 14 DW1 485
Decouple Register 15 DW0 486
Decouple Register 15 DW1 487
DE Misc Interrupt Definition 488
DE Port Interrupt Definition 490

Device 2 Control 492
Device Enable 493
Device Identification 495
DFSDONE 496
DFSM 497
DISPIO_CR_TX_BMU_CR0 500
Display CSR Program 501
Display Message Forward Status Register 513
DOUBLE_BUFFER_CTL 515
DP_TP_CTL 517
DP_TP_STATUS 521
DPLL_CFGCR1 525
DPLL_CFGCR2 526
DPLL_CTRL1 528
DPLL_CTRL2 532
DPLL_STATUS 535
DPST_BIN 538
DPST_CTL 539
DPST_GUARD 541
Driver Media Force Wake Ack 542
Driver Render Force Wake Ack 543
DS Invocation Counter 544
DSSM 545
DX9 Constants Not Consumed By RCS 547
DX9 Constants Prsed By RCS 548
ECO reg 1 549
ECO Reserved 550
Element Descriptor Register 551
EMRR Mask LSB 554
EMRR Mask MSB 555
Error Identity Register 556
Error Mask Register 558
Error Reporting Register 560

Error Status Register 562
EU_GRF_CLEAR 563
EU_STALL PER SUBSLICE 564
EU Mask Programming 565
EU NOT IDLE PER SUBSLICE 571
EU PAIR 0 PFET control register with lock 572
EU PAIR 0 PGFET control register with lock 574
EU PAIR 0 Power Context Save request 576
EU PAIR 0 Power Down FSM control register with lock 577
EU PAIR 0 Power Gate Control Request 580
EU PAIR 0 Power on FSM control register with lock 581
EU PAIR 1 PFET control register with lock 583
EU PAIR 1 Power Context Save request 585
EU PAIR 1 Power Down FSM control register with lock 586
EU PAIR 1 Power Gate Control Request 589
EU PAIR 1 Power on FSM control register with lock 590
EU PAIR 2 PGFET control register with lock 592
EU PAIR 2 Power Context Save request 594
EU PAIR 2 power Down FSM control register with lock 595
EU PAIR 2 Power Gate Control Request 598
EU PAIR 2 Power on FSM control register with lock 599
EU PAIR 3 PFET control register with lock 601
EU PAIR 3 Power Context Save request 603
EU PAIR 3 Power Down FSM control register with lock 604
EU PAIR 3 Power Gate Control Request 607
EU PAIR 3 Power on FSM control register with lock 608
Event selection and base counters 610
Event Selection and Base Counters1 613
Execlist 0 Contents 614
Execlist 1 Contents 615
Execlist Status 616
Execlist Submit Port Register 619
Execute Condition Code Register 621

FAULT_TLB_RD_DATA0 Register 623
FAULT_TLB_RD_DATA1 Register 624
Fault Mode Control 625
Fault Mode Control 626
Fault Switch Out 627
FBC_CFB_BASE 628
FBC_CTL 629
FBC_RT_BASE_ADDR_REGISTER 632
FBC_RT_BASE_ADDR_REGISTER_UPPER 633
FBC LLC Config Read Control Register 634
Fence Control Register 635
FF Performance 637
First Buffer Size and Start 639
Flexible EU Event Control 0 641
Flexible EU Event Control 1 642
Flexible EU Event Control 2 643
Flexible EU Event Control 3 644
Flexible EU Event Control 4 645
Flexible EU Event Control 5 646
Flexible EU Event Control 6 647
FORCE_TO_NONPRIV 648
Frame Buffer Cache Definition Register 654
Frame count and Draw call number 656
FUSE_STATUS 657
GAB Arbitration Programmable 659
GAB LRA 0 660
GAB LRA 1 662
GAB unit Control Register 663
GAC_GAM Arbitration Counters Register 0 664
GAC_GAM Arbitration Counters Register 1 665
GAC_GAM R Arbitration Register 0 666
GAC_GAM R Arbitration Register 1 667
GAC_GAM R Arbitration Register 2 668

GAC_GAM R Arbitration Register 3 669
GAC_GAM RO Arbitration Register 0 670
GAC_GAM RO Arbitration Register 1 671
GAC_GAM RO Arbitration Register 2 672
GAC_GAM RO Arbitration Register 3 673
GAC_GAM WR Arbitration Register 0 674
GAC_GAM WR Arbitration Register 1 675
GAC_GAM WR Arbitration Register 2 676
GAC_GAM WR Arbitration Register 3 677
GAFS MAX URB READ EVENT 678
GAM and SA Communication Register 679
GAM Context Save 682
GAM Context Save Register 683
Gam Fub Done1 Lookup Register 684
Gam Fub Done Lookup Register 685
GAMMA_MODE 686
GAM Put Delay 688
GAMT_DONE Register 689
GAMT_ECO_REG_RO_IA 690
GAMT_ECO_REG_RW_IA 691
GAMT Arbiter Mode Control 693
GAM Virtualization Enable Register 696
GAMW_ECO_BUS_RO_IA 697
GAMW_ECO_BUS_RW_IA 698
GAMW_ECO_DEV_RO_IA 699
GAMW_ECO_DEV_RW_IA 700
GAMW Power Context Save 701
Gather Constants Not Consumed By RCS 703
GDR Per Client Write Drop Enables 704
GDR Per Client Write Drop Enables-2 705
GDR Write Drop 706
General Purpose Register 707
GFX Arbiter Client Priority Control 711

GFX Context Element Descriptor (High Part) 713
GFX Context Element Descriptor (Low Part) 714
GFX Fault Counter 716
GFX Fixed Counter 717
GFX PDP0/PML4/PASID Descriptor (High Part) 718
GFX PDP0/PML4/PASID Descriptor (Low Part) 719
GFX PDP1 Descriptor Register (High Part) 720
GFX PDP1 Descriptor Register (Low Part) 721
GFX PDP2 Descriptor Register (High Part) 722
GFX PDP2 Descriptor Register (Low Part) 723
GFX PDP3 Descriptor Register (High Part) 724
GFX PDP3 Descriptor Register (Low Part) 725
Global Invalidation Register 726
Global System Interrupt Routine 727
GMBUS0 728
GMBUS1 730
GMBUS2 733
GMBUS3 736
GMBUS4 737
GMBUS5 738
GMCH Graphics Control 739
Go Protocol GAM Request 742
GPA to HPA Translation Request 746
GPA value for GPA to HPA Translation 749
GPGPU Context Restore Request To TDL 750
GPGPU Context Save Request To TDL 752
GPGPU Dispatch Dimension X 753
GPGPU Dispatch Dimension Y 754
GPGPU Dispatch Dimension Z 755
GPIO_CTL 756
GPM POWERGATE LICENSE REQUEST 759
GP Thread Time 760
GPU_Ticks_Counter 761

Graphics Device Reset Control 762
Graphics Memory Fence Table Register 765
Graphics Memory Range Address 769
Graphics MOCS Register0 771
Graphics MOCS Register1 773
Graphics MOCS Register2 775
Graphics MOCS Register3 777
Graphics MOCS Register4 779
Graphics MOCS Register5 781
Graphics MOCS Register6 783
Graphics MOCS Register7 785
Graphics MOCS Register8 787
Graphics MOCS Register9 789
Graphics MOCS Register10 791
Graphics MOCS Register11 793
Graphics MOCS Register12 795
Graphics MOCS Register13 797
Graphics MOCS Register14 799
Graphics MOCS Register15 801
Graphics MOCS Register16 803
Graphics MOCS Register17 805
Graphics MOCS Register18 807
Graphics MOCS Register19 809
Graphics MOCS Register20 811
Graphics MOCS Register21 813
Graphics MOCS Register22 815
Graphics MOCS Register23 817
Graphics MOCS Register24 819
Graphics MOCS Register25 821
Graphics MOCS Register26 823
Graphics MOCS Register27 825
Graphics MOCS Register28 827
Graphics MOCS Register29 829

Graphics MOCS Register30 831
Graphics MOCS Register31 833
Graphics MOCS Register32 835
Graphics MOCS Register33 837
Graphics MOCS Register34 839
Graphics MOCS Register35 841
Graphics MOCS Register36 843
Graphics MOCS Register37 845
Graphics MOCS Register38 847
Graphics MOCS Register39 849
Graphics MOCS Register40 851
Graphics MOCS Register41 853
Graphics MOCS Register42 855
Graphics MOCS Register43 857
Graphics MOCS Register44 859
Graphics MOCS Register45 861
Graphics MOCS Register46 863
Graphics MOCS Register47 865
Graphics MOCS Register48 867
Graphics MOCS Register49 869
Graphics MOCS Register50 871
Graphics MOCS Register51 873
Graphics MOCS Register52 875
Graphics MOCS Register53 877
Graphics MOCS Register54 879
Graphics MOCS Register55 881
Graphics MOCS Register56 883
Graphics MOCS Register57 885
Graphics MOCS Register58 887
Graphics MOCS Register59 889
Graphics MOCS Register60 891
Graphics MOCS Register61 893
Graphics MOCS Register62 895

Graphics MOCS Register63 897
Graphics Mode Register 899
Graphics System Event 902
Graphics Translation Table Memory Mapped Range Address 903
GSA_AUDIO_BDF 905
GSA_TOUCH_BDF 906
GS Invocation Counter 907
GS Primitives Counter 908
GT4 Mode Control Register 909
GTC_CTL 910
GTC_DDA_M 911
GTC_DDA_N 912
GTC_IIR 913
GTC_IMR 914
GTC_LIVE 915
GTC_PORT_CTL 916
GTC_PORT_MISC 918
GTC_PORT_TX_CURR 920
GTC_PORT_TX_PREV 921
GTFORCEAWAKE 922
GT Function Level Reset Control Message 923
GT Interrupt 0 Definition 924
GT Interrupt 1 Definition 926
GT Interrupt 2 Definition 928
GT Interrupt 3 Definition 930
GTI PFET control register with lock 932
GTI Power Gate Control Request 935
GT Mode Register 936
GTSCRATCH 939
GTSP0 940
GTSP1 941
GTSP2 942
GTSP3 943

GTSP4 944
GTSP5 945
GTSP6 946
GTSP7 947
GTT Cache Enable 948
Hardware Scratch Read Write 949
Hardware Status Mask Register 950
Hardware Status Page Address Register 952
HCP Bitstream Output Minimal Size Padding Count Report Register 953
HCP CABAC Status 954
HCP Decode Status 956
HCP Frame BitStream BIN Count 957
HCP Frame Motion Comp Miss Count 958
HCP Frame Motion Comp Read Count 959
HCP Frame Performance Count 960
HCP Image Status Control 961
HCP Image Status Mask 963
HCP Memory Latency Count1 964
HCP Memory Latency Count2 965
HCP Memory Latency Count3 966
HCP Memory Latency Count4 967
HCP Memory Latency Count5 968
HCP Memory Latency Count6 969
HCP Picture Checksum cIdx0 970
HCP Picture Checksum cIdx1 971
HCP Picture Checksum cIdx2 972
HCP Qp Status Count 973
HCP Reported Bitstream Output CABAC Bin Count Register 974
HCP Slice Performance Count 975
HCP Unit Done 976
HDC TLB REQUEST CONTROL REGISTER 977
HDPORT_STATE 978
Header Type 981

HEVC GAM Slave Counter High part 982
HEVC GAM Slave Counter Low part 983
HEVC Local APIC Retry Vector 984
HEVC Microcontroller Header Info 985
HS Invocation Counter 986
IA32 MTRR FIX4K_C0000 High 987
IA32 MTRR FIX4K_C0000 Low 988
IA32 MTRR FIX4K_C8000 High 989
IA32 MTRR FIX4K_C8000 Low 990
IA32 MTRR FIX4K_D0000 High 991
IA32 MTRR FIX4K_D0000 Low 992
IA32 MTRR FIX4K_D8000 High 993
IA32 MTRR FIX4K_D8000 Low 994
IA32 MTRR FIX4K_E0000 High 995
IA32 MTRR FIX4K_E0000 Low 996
IA32 MTRR FIX4K_E8000 High 997
IA32 MTRR FIX4K_E8000 Low 998
IA32 MTRR FIX4K_F0000 High 999
IA32 MTRR FIX4K_F0000 Low 1000
IA32 MTRR FIX4K_F8000 High 1001
IA32 MTRR FIX4K_F8000 Low 1002
IA32 MTRR FIX16K_80000 High 1003
IA32 MTRR FIX16K_80000 Low 1004
IA32 MTRR FIX16K_A0000 High 1005
IA32 MTRR FIX16K_A0000 Low 1006
IA32 MTRR FIX64K_00000 High 1007
IA32 MTRR FIX64K_00000 Low 1008
IA32 MTRR PHYSBASE0 High 1009
IA32 MTRR PHYSBASE0 Low 1010
IA32 MTRR PHYSBASE1 High 1011
IA32 MTRR PHYSBASE1 Low 1012
IA32 MTRR PHYSBASE2 High 1013
IA32 MTRR PHYSBASE2 Low 1014

IA32 MTRR PHYSBASE3 High 1015
IA32 MTRR PHYSBASE3 Low 1016
IA32 MTRR PHYSBASE4 High 1017
IA32 MTRR PHYSBASE4 Low 1018
IA32 MTRR PHYSBASE5 High 1019
IA32 MTRR PHYSBASE5 Low 1020
IA32 MTRR PHYSBASE6 High 1021
IA32 MTRR PHYSBASE6 Low 1022
IA32 MTRR PHYSBASE7 High 1023
IA32 MTRR PHYSBASE7 Low 1024
IA32 MTRR PHYSBASE8 High 1025
IA32 MTRR PHYSBASE8 Low 1026
IA32 MTRR PHYSBASE9 High 1027
IA32 MTRR PHYSBASE9 Low 1028
IA32 MTRR PHYSMASK0 High 1029
IA32 MTRR PHYSMASK0 Low 1030
IA32 MTRR PHYSMASK1 High 1031
IA32 MTRR PHYSMASK1 Low 1032
IA32 MTRR PHYSMASK2 High 1033
IA32 MTRR PHYSMASK2 Low 1034
IA32 MTRR PHYSMASK3 High 1035
IA32 MTRR PHYSMASK3 Low 1036
IA32 MTRR PHYSMASK4 High 1037
IA32 MTRR PHYSMASK4 Low 1038
IA32 MTRR PHYSMASK5 High 1039
IA32 MTRR PHYSMASK5 Low 1040
IA32 MTRR PHYSMASK6 High 1041
IA32 MTRR PHYSMASK6 Low 1042
IA32 MTRR PHYSMASK7 High 1043
IA32 MTRR PHYSMASK7 Low 1044
IA32 MTRR PHYSMASK8 High 1045
IA32 MTRR PHYSMASK8 Low 1046
IA32 MTRR PHYSMASK9 High 1047

IA32 MTRR PHYSMASK9 Low 1048
IA Vertices Count 1049
IDI Cacheable Register 1050
IDI Control register 1052
IDI HASH Mask Register 1055
IDI Look up Register 1056
IDILook up Table register 1061
IDI MESSAGES 1063
IDI Self Snoop Register 1065
Idle Switch Delay 1068
Indirect Context Offset Pointer 1069
Indirect Context Pointer 1071
INF unit Level Clock Gating Control 9560 1074
Instruction Parser Mode Register 1076
Internal GAM State 1078
Interrupt Line 1079
Interrupt Mask Register 1080
Interrupt Pin 1081
I/O Base Address 1082
IPC PER SUBSLICE 1083
KCR GAM slave counter High part 1084
KCR GAM slave counter Low part 1085
L3 Bank Status 1086
L3 Control Register 1088
L3 Control Register1 1090
L3 LRA 0 1093
L3 LRA 0 GPGPU 1094
L3 LRA 1 1095
L3 LRA 1 GPGPU 1096
L3 LRA 2 1097
L3 LRA 2 GPGPU 1098
L3 LRA 0 3D 1099
L3 LRA 1 3D 1100

L3 LRA 2 3D 1101
L3 LRA 2 3D 1102
L3 SLM Register 1103
L3 SQC register 4 1104
L3 SQC registers 1 1107
L3 SQC registers 2 1111
L3 SQC registers 3 1114
LBCF config save msg 1119
LBS config bits 1120
LCPLL1_CTL 1121
LCPLL2_CTL 1122
LINKM 1123
LINKN 1124
LNCF config save msg 1125
LNCF MOCS Register 0 1126
LNCF MOCS Register 0 1128
LNCF MOCS Register 1 1130
LNCF MOCS Register 1 1131
LNCF MOCS Register 2 1132
LNCF MOCS Register 2 1133
LNCF MOCS Register 3 1134
LNCF MOCS Register 3 1135
LNCF MOCS Register 4 1136
LNCF MOCS Register 4 1137
LNCF MOCS Register 5 1138
LNCF MOCS Register 5 1139
LNCF MOCS Register 6 1140
LNCF MOCS Register 6 1141
LNCF MOCS Register 7 1142
LNCF MOCS Register 7 1143
LNCF MOCS Register 8 1144
LNCF MOCS Register 8 1145
LNCF MOCS Register 9 1146

LNCF MOCS Register 9 1147
LNCF MOCS Register 10 1148
LNCF MOCS Register 10 1149
LNCF MOCS Register 11 1150
LNCF MOCS Register 11 1151
LNCF MOCS Register 12 1152
LNCF MOCS Register 12 1153
LNCF MOCS Register 13 1154
LNCF MOCS Register 13 1155
LNCF MOCS Register 14 1156
LNCF MOCS Register 14 1157
LNCF MOCS Register 15 1158
LNCF MOCS Register 15 1159
LNCF MOCS Register 16 1160
LNCF MOCS Register 16 1161
LNCF MOCS Register 17 1162
LNCF MOCS Register 17 1163
LNCF MOCS Register 18 1164
LNCF MOCS Register 18 1165
LNCF MOCS Register 19 1166
LNCF MOCS Register 19 1167
LNCF MOCS Register 20 1168
LNCF MOCS Register 20 1169
LNCF MOCS Register 21 1170
LNCF MOCS Register 21 1171
LNCF MOCS Register 22 1172
LNCF MOCS Register 22 1173
LNCF MOCS Register 23 1174
LNCF MOCS Register 23 1175
LNCF MOCS Register 24 1176
LNCF MOCS Register 24 1177
LNCF MOCS Register 25 1178
LNCF MOCS Register 25 1179

LNCF MOCS Register 26 1180
LNCF MOCS Register 26 1181
LNCF MOCS Register 27 1182
LNCF MOCS Register 27 1183
LNCF MOCS Register 28 1184
LNCF MOCS Register 28 1185
LNCF MOCS Register 29 1186
LNCF MOCS Register 29 1187
LNCF MOCS Register 30 1188
LNCF MOCS Register 30 1189
LNCF MOCS Register 31 1190
LNCF MOCS Register 31 1191
LNCF Render config save msg 1192
Load Indirect Base Vertex 1193
Load Indirect Instance Count 1194
Load Indirect Start Instance 1195
Load Indirect Start Vertex 1196
Load Indirect Vertex Count 1197
LPFC control register 1198

Active Head Pointer Register

ACTHD - Active Head Pointer Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	02074h-02077h
Name:	Active Head Pointer Register
ShortName:	ACTHD_RCSUNIT
Address:	12074h-12077h
Name:	Active Head Pointer Register
ShortName:	ACTHD_VCSUNIT0
Address:	1A074h-1A077h
Name:	Active Head Pointer Register
ShortName:	ACTHD_VECSUNIT
Address:	1C074h-1C077h
Name:	Active Head Pointer Register
ShortName:	ACTHD_VCSUNIT1
Address:	22074h-22077h
Name:	Active Head Pointer Register
ShortName:	ACTHD_BCSUNIT

This register contains the address details of the data dword being parsed by command streamer.

- When the commands are being executed from a batch buffer this register contains the Dword aligned Graphics Memory Address.
- When the commands are being executed from a ring buffer this register contains the Dword aligned offset in to the ring buffer (offset from Ring Buffer start address).

DWord	Bit	Description
0	31:2	**Head Pointer** Format: GraphicsAddress[31:2] • When the commands are being executed from a batch buffer this register contains the Dword aligned Graphics Memory Address. • When the commands are being executed from a ring buffer this register contains the Dword aligned offset in to the ring buffer (offset from Ring Buffer start address).

ACTHD - Active Head Pointer Register			
	1:0	**Reserved**	
		Format:	MBZ

Advanced Scheduler Reset Request Messages

ASSRREQ - Advanced Scheduler Reset Request Messages

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0810Ch

Hardware (CS, VCS) initiated Advanced Scheduler reset request messages.

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask To write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:6	**Reserved** Access: RO Reserved
	5	**SFC1 gracefull reset request message (2nd Vbox)** Access: R/W Set SFC1 gracefull Reset Request Message for 2nd Vbox: '1' : cmsfc Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : cmsfc Reset Not Requested
	4	**SFC0 gracefull reset request message (1st Vbox)** Access: R/W Set SFC0 gracefull Reset Request Message for 1st Vbox: '1' : cmsfc Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : cmsfc Reset Not Requested
	3	**VINunit cmfxrst reset request message (2nd Vbox)** Access: R/W Set CMFX Reset Request Message from the VINunit in 2nd Vbox: '1' : CMFX Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : CMFX Reset Not Requested

ASSRREQ - Advanced Scheduler Reset Request Messages

	2	**VINunit cmfxrst Reset Request message** Access: R/W Set CMFX Reset Request Message from the VINunit: '1' : CMFX Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : CMFX Reset Not Requested
	1	**Render AS Reset Request Message** Access: R/W Set Render AS Reset Request Message from the CSunit: '1' : Render AS Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : Render AS Reset Not Requested
	0	**Media AS Reset Request Message** Access: R/W Set Media AS Reset Request Message from the VCSunit: '1' : Media AS Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : Media AS Reset Not Requested

Aggregate_Perf_Counter_A31

OAPERF_A31 - Aggregate_Perf_Counter_A31		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	028F8h	
This register reflects the count value of the OA Performance counter A31		
DWord	**Bit**	**Description**
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A32

OAPERF_A32 - Aggregate_Perf_Counter_A32

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02900h

This register reflects the count value of the OA Performance counter A32

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A33

OAPERF_A33 - Aggregate_Perf_Counter_A33	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02904h

This register reflects the count value of the OA Performance counter A33

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A34

OAPERF_A34 - Aggregate_Perf_Counter_A34	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02908h

This register reflects the count value of the OA Performance counter A34

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A35

OAPERF_A35 - Aggregate_Perf_Counter_A35	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0290Ch

This register reflects the count value of the OA Performance counter A35

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A0

OAPERF_A0 - Aggregate Perf Counter A0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02800h

This register reflects the count value of the OA Performance counter A0.

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Value	Name
0x00000000	**[Default]**
[0x00000001-0xFFFFFFFF]	

Aggregate Perf Counter A0 Upper DWord

OAPERF_A0_UPPER - Aggregate Perf Counter A0 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02804h

This register enables the current live value of performance counter A0 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A1

OAPERF_A1 - Aggregate Perf Counter A1	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02808h

This register reflects the count value of the OA Performance counter A1. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A1 Upper DWord

OAPERF_A1_UPPER - Aggregate Perf Counter A1 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0280Ch

This register enables the current live value of performance counter A1 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A2

OAPERF_A2 - Aggregate Perf Counter A2	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02810h

This register reflects the count value of the OA Performance counter A2. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A2 Upper DWord

OAPERF_A2_UPPER - Aggregate Perf Counter A2 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02814h

This register enables the current live value of performance counter A2 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A3

OAPERF_A3 - Aggregate Perf Counter A3	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02818h

This register reflects the count value of the OA Performance counter A3. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A3 Upper DWord

OAPERF_A3_UPPER - Aggregate Perf Counter A3 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0281Ch

This register enables the current live value of performance counter A3 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A4

OAPERF_A4 - Aggregate Perf Counter A4	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02820h

This register reflects the count value of the OA Performance counter A4. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A4 Lower DWord Free

OAPERF_A4_LOWER_FREE - Aggregate Perf Counter A4 Lower DWord Free	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02960h

This register counts the same event as counter A4 however is not affected by context ID or other conditions that prevent A4 from incrementing. This counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted used this register.

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A4 Upper DWord

OAPERF_A4_UPPER - Aggregate Perf Counter A4 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02824h

This register enables the current live value of performance counter A4 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A4 Upper DWord Free

OAPERF_A4_UPPER_FREE - Aggregate Perf Counter A4 Upper DWord Free

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02964h

This register counts the same event as counter A4 however is not affected by context ID or other conditions that prevent A4 from incrementing. This counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted using this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A5

OAPERF_A5 - Aggregate Perf Counter A5	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02828h

This register reflects the count value of the OA Performance counter A5. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A5 Upper DWord

OAPERF_A5_UPPER - Aggregate Perf Counter A5 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0282Ch

This register enables the current live value of performance counter A5 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A6

OAPERF_A6 - Aggregate Perf Counter A6	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02830h

This register reflects the count value of the OA Performance counter A6. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A6 Lower DWord Free

OAPERF_A6_LOWER_FREE - Aggregate Perf Counter A6 Lower DWord Free	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02968h

This register counts the same event as counter A6 however is not affected by context ID or other conditions that prevent A6 from incrementing. his counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted used this register.

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A6 Upper DWord

OAPERF_A6_UPPER - Aggregate Perf Counter A6 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02834h

This register enables the current live value of performance counter A6 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A6 Upper DWord Free

OAPERF_A6_UPPER_FREE - Aggregate Perf Counter A6 Upper DWord Free

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0296Ch

This register counts the same event as counter A6 however is not affected by context ID or other conditions that prevent A6 from incrementing. his counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted using this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A7

OAPERF_A7 - Aggregate Perf Counter A7	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02838h

This register reflects the count value of the OA Performance counter A7. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A7 Upper DWord

OAPERF_A7_UPPER - Aggregate Perf Counter A7 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0283Ch

This register enables the current live value of performance counter A7 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A8

OAPERF_A8 - Aggregate Perf Counter A8	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02840h

This register reflects the count value of the OA Performance counter A8. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A8 Upper DWord

OAPERF_A8_UPPER - Aggregate Perf Counter A8 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02844h

This register enables the current live value of performance counter A8 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A9

OAPERF_A9 - Aggregate Perf Counter A9	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02848h

This register reflects the count value of the OA Performance counter A9. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A9 Upper DWord

OAPERF_A9_UPPER - Aggregate Perf Counter A9 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0284Ch

This register enables the current live value of performance counter A9 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A10

OAPERF_A10 - Aggregate Perf Counter A10	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02850h

This register reflects the count value of the OA Performance counter A10. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A10 Upper DWord

OAPERF_A10_UPPER - Aggregate Perf Counter A10 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02854h

This register enables the current live value of performance counter A10 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A11

OAPERF_A11 - Aggregate Perf Counter A11	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02858h

This register reflects the count value of the OA Performance counter A11. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A11 Upper DWord

OAPERF_A11_UPPER - Aggregate Perf Counter A11 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0285Ch

This register enables the current live value of performance counter A11 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A12

OAPERF_A12 - Aggregate Perf Counter A12	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02860h

This register reflects the count value of the OA Performance counter A12. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A12 Upper DWord

OAPERF_A12_UPPER - Aggregate Perf Counter A12 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02864h

This register enables the current live value of performance counter A12 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A13

OAPERF_A13 - Aggregate Perf Counter A13	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02868h

This register reflects the count value of the OA Performance counter A13. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A13 Upper DWord

OAPERF_A13_UPPER - Aggregate Perf Counter A13 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0286Ch

This register enables the current live value of performance counter A13 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A14

OAPERF_A14 - Aggregate Perf Counter A14	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02870h

This register reflects the count value of the OA Performance counter A14. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A14 Upper DWord

OAPERF_A14_UPPER - Aggregate Perf Counter A14 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02874h

This register enables the current live value of performance counter A14 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A15

OAPERF_A15 - Aggregate Perf Counter A15		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	02878h	
This register reflects the count value of the OA Performance counter A15. DefaultValue="00000000h"		
DWord	**Bit**	**Description**
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A15 Upper DWord

OAPERF_A15_UPPER - Aggregate Perf Counter A15 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0287Ch

This register enables the current live value of performance counter A15 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A16

OAPERF_A16 - Aggregate Perf Counter A16	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02880h

This register reflects the count value of the OA Performance counter A16. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A16 Upper DWord

OAPERF_A16_UPPER - Aggregate Perf Counter A16 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02884h

This register enables the current live value of performance counter A16 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A17

OAPERF_A17 - Aggregate Perf Counter A17	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02888h

This register reflects the count value of the OA Performance counter A17. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A17 Upper DWord

OAPERF_A17_UPPER - Aggregate Perf Counter A17 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0288Ch

This register enables the current live value of performance counter A17 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A18

OAPERF_A18 - Aggregate Perf Counter A18	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02890h

This register reflects the count value of the OA Performance counter A9. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A18 Upper DWord

OAPERF_A18_UPPER - Aggregate Perf Counter A18 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02894h

This register enables the current live value of performance counter A18 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A19

OAPERF_A19 - Aggregate Perf Counter A19	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02898h

This register reflects the count value of the OA Performance counter A19. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A19 Lower DWord Free

OAPERF_A19_LOWER_FREE - Aggregate Perf Counter A19 Lower DWord Free	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02970h

This register counts the same event as counter A19 however is not affected by context ID or other conditions that prevent A19 from incrementing. This counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted used this register.

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A19 Upper DWord

OAPERF_A19_UPPER - Aggregate Perf Counter A19 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0289Ch

This register enables the current live value of performance counter A19 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A19 Upper DWord Free

OAPERF_A19_UPPER_FREE - Aggregate Perf Counter A19 Upper DWord Free

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02974h

This register counts the same event as counter A19 however is not affected by context ID or other conditions that prevent A19 from incrementing. This counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted using this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A20

OAPERF_A20 - Aggregate Perf Counter A20	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028A0h

This register reflects the count value of the OA Performance counter A20. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A20 Lower DWord Free

OAPERF_A20_LOWER_FREE - Aggregate Perf Counter A20 Lower DWord Free	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02978h

This register counts the same event as counter A20 however is not affected by context ID or other conditions that prevent A20 from incrementing. This counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted used this register.

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A20 Upper DWord

OAPERF_A20_UPPER - Aggregate Perf Counter A20 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028A4h

This register enables the current live value of performance counter A20 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A20 Upper DWord Free

OAPERF_A20_UPPER_FREE - Aggregate Perf Counter A20 Upper DWord Free

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0297Ch

This register counts the same event as counter A20 however is not affected by context ID or other conditions that prevent A20 from incrementing. his counter is free running, always enabled and counting irrespective of OA enabled or disabled. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is counted using this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A21

OAPERF_A21 - Aggregate Perf Counter A21	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028A8h

This register reflects the count value of the OA Performance counter A21. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A21 Upper DWord

OAPERF_A21_UPPER - Aggregate Perf Counter A21 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028ACh

This register enables the current live value of performance counter A21 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A22

OAPERF_A22 - Aggregate Perf Counter A22	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028B0h

This register reflects the count value of the OA Performance counter A22. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A22 Upper DWord

OAPERF_A22_UPPER - Aggregate Perf Counter A22 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028B4h

This register enables the current live value of performance counter A22 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A23

OAPERF_A23 - Aggregate Perf Counter A23	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028B8h

This register reflects the count value of the OA Performance counter A23. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A23 Upper DWord

OAPERF_A23_UPPER - Aggregate Perf Counter A23 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028BCh

This register enables the current live value of performance counter A23 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A24

OAPERF_A24 - Aggregate Perf Counter A24	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028C0h

This register reflects the count value of the OA Performance counter A24. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A24 Upper DWord

OAPERF_A24_UPPER - Aggregate Perf Counter A24 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028C4h

This register enables the current live value of performance counter A24 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A25

OAPERF_A25 - Aggregate Perf Counter A25	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028C8h

This register reflects the count value of the OA Performance counter A25. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A25 Upper DWord

OAPERF_A25_UPPER - Aggregate Perf Counter A25 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028CCh

This register enables the current live value of performance counter A25 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A26

OAPERF_A26 - Aggregate Perf Counter A26	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028D0h

This register reflects the count value of the OA Performance counter A26. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A26 Upper DWord

OAPERF_A26_UPPER - Aggregate Perf Counter A26 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028D4h

This register enables the current live value of performance counter A26 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A27

OAPERF_A27 - Aggregate Perf Counter A27	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028D8h

This register reflects the count value of the OA Performance counter A27. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A27 Upper DWord

OAPERF_A27_UPPER - Aggregate Perf Counter A27 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028DCh

This register enables the current live value of performance counter A27 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A28

OAPERF_A28 - Aggregate Perf Counter A28	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028E0h

This register reflects the count value of the OA Performance counter A28. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A28 Upper DWord

OAPERF_A28_UPPER - Aggregate Perf Counter A28 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028E4h

This register enables the current live value of performance counter A28 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A29

OAPERF_A29 - Aggregate Perf Counter A29	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028E8h

This register reflects the count value of the OA Performance counter A29. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A29 Upper DWord

OAPERF_A29_UPPER - Aggregate Perf Counter A29 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028ECh

This register enables the current live value of performance counter A29 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A30

OAPERF_A30 - Aggregate Perf Counter A30	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028F0h

This register reflects the count value of the OA Performance counter A30. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A30 Upper DWord

OAPERF_A30_UPPER - Aggregate Perf Counter A30 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028F4h

This register enables the current live value of performance counter A30 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A31 Upper DWord

OAPERF_A31_UPPER - Aggregate Perf Counter A31 Upper DWord

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028FCh

This register enables the current live value of performance counter A31 to be read. Please refer to the detailed description of the aggregating performance counters in the Aggregating Counters section for more details on what event is reported via this register.

DWord	Bit	Description
0	31:8	**Reserved** Format: PBC
	7:0	**Upper Value** Format: U8 This 8-bit field returns bits 39:32 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

All Engine Fault Register

FAULT_REG - All Engine Fault Register

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04094h

DWord	Bit	Description
0	31:1	**All Engine Fault Register** Default Value: 00000000000000000000000000000000b Access: R/W Bit[31:15]: Reserved. Bit[14:12]: Engine ID: 000b - GFX. 001b - MFX0. 010b - MFX1. 011b - VEBX. 100b - BLT. 110b - Reserved Bit[11]: Reserved. Bit[10:3]: SRCID of Fault. This is the Source ID of the unit that requested the cycle that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW. Bit[2:1]: Fault Type (GFX_FT): Type of Fault recorded: 00b - Invalid PTE Fault. 01b - Invalid PDE Fault. 10b - Invalid PDPE Fault. 11b - Invalid PML4E Fault. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW. All bits are only valid with bit[0]=1.
	0	**Valid Bit** Default Value: 0b Access: R/W This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which also clears the other fields.

ARB_CTL

ARB_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0E661056 [KBL]
Access:	R/W
Size (in bits):	32
Address:	45000h-45003h
Name:	Display Arbitration Control 1
ShortName:	ARB_CTL
Power:	PG0
Reset:	soft

DWord	Bit	Description
0	31	**FBC Memory Wake** Setting this bit allows FBC compressed write requests to wake memory. Value: 1b — Name: Wake On Value: 0b — Name: Wake Off
	30	**Reserved**
	29	**Reserved**
	28:26	**HP Queue Watermark** Default Value: 011b 4 entries The value in this register indicates the number of entries the high priority queue should have before it can be read. The value is zero based.
	25:24	**LP Write Request Limit** The value in this register indicates the maximum number of back to back LP write requests that will be accepted from a single client before re-arbitrating. Value: 00b — Name: 1 Value: 01b — Name: 2 Value: 10b — Name: 4 **[Default]** Value: 11b — Name: 8
	23:20	**TLB Request Limit** The value in this register indicates the maximum number of TLB requests that can be made in an arbitration loop. Zero is not a valid programming. Value: 0110b — Name: 6 **[Default]**

ARB_CTL

	[1,15]

19:16	**TLB Request InFlight Limit** The value in this register indicates the maximum number of TLB (or VTd) requests that can be in flight at any given time. Zero is not a valid programming. **Value** / **Name** 0110b — 6 **[Default]** [1,15]
15	**FBC Watermark Disable** Setting this bit disables the FBC watermarks. **Value** / **Name** 0b — Enable 1b — Disable
14:13	**Tiled Address Swizzling** DRAM configuration registers show if memory address swizzling is needed. **Value** / **Name** / **Description** 00b — No Display — No display request address swizzling 01b — Reserved — Address bit[6] swizzling for tiled surfaces is not used 10b — Reserved — 11b — Reserved —
12:8	**HP Page Break Limit** The value in this register represents the maximum number of page breaks allowed in a HP request chain. Zero is not a valid programming. **Value** / **Name** 10000b — 16 **[Default]** [1,31]
7	**Reserved**
6:0	**HP Data Request Limit** The value in this register represents the maximum number of cachelines allowed in a HP request chain. **Value** / **Name** 1010110b — 86 **[Default]** [1,127] **Restriction** Restriction: This value must always be programmed greater than 8.

ARB_CTL2

ARB_CTL2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x20000600
Access:	R/W
Size (in bits):	32
Address:	45004h-45007h
Name:	Display Arbitration Control 2
ShortName:	ARB_CTL2
Power:	PG0
Reset:	soft

DWord	Bit	Description
0	31	**Reserved**
	30	**Reserved** Format: MBZ
	29:28	**LP WD Write Request Limit** The value in this register indicates the maximum number of back to back LP write requests that will be accepted from WD before re-arbitrating. 00b: 1 01b: 2 10b: 4 **[Default]** 11b: 8
	27:25	**Reserved** Format: MBZ
	24:20	**Reserved** Format: MBZ
	19:16	**Reserved** Format: MBZ
	15	**Reserved** Format: MBZ
	14	**Reserved** Format: MBZ
	13	**Reserved**

ARB_CTL2

12 — Arbiter Trickle Feed Allow On HP Request

Description	Project
If enabled, Arbiter will allow trickle feed request from all clients if any of the client sends a high priority request	
This field must be kept at default value.	KBL:*:A, KBL:*:B

Value	Name
0b	Disable **[Default]**
1b	Enable

11 — Reserved

10:9 — Inflight LP Read Request Limit

The value in this register represents the maximum number of LP read request transactions that can be inflight at any given time.

Value	Name
00b	1 LP
01b	2 LP
10b	3 LP
11b	4 LP **[Default]**

8 — Reserved

Format:	MBZ

7 — Reserved

6 — Reserved

Format:	MBZ

5:4 — Inflight HP Read Request Limit

The value in this register represents the maximum number of HP read request transactions that can be inflight at any given time.

Value	Name
00b	128 HP
01b	64 HP
10b	32 HP
11b	16 HP

ARB_CTL2

	3	**Enable IPC** Enables the Isochronous Priority Control. If enabled, Display sends demoted requests once the transition watermark is reached. If transition watermark is not enabled, Display sends demoted requests when the display buffer is full. **Value** / **Name** 0b – Disable 1b – Enable
	2	**Reserved** Format: MBZ
	1:0	**RTID FIFO Watermark** The value in this register represents the watermark value for the RTID FIFO. HP transactions will start only when the FIFO level is above or equal the watermark. **Value** / **Name** 00b – 8 RTIDs 01b – 16 RTIDs 10b – 32 RTIDs 11b – Reserved

Arbiter Control Register

GARBCNTLREG - Arbiter Control Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x29124100 [KBL]
Size (in bits):	32
Address:	0B004h

DWord	Bit	Description
0	31	**Reserved**
	30	**Disables hashing function** Access: R/W Disables hashing function to generate bank_id[1:0] for L3$ bank accessing, and forces the use of address[7:6] for bank_id[1:0]. 0: (default) Hash function enabled to generate L3$ bank IDs. 1: L3$ address[7:6] used as L3$ bank IDs. Incf_csr_l3bankidhashdis. (This bit needs to set corresponding bit lpfcon_csr_l3bankidhashdis in LPFC.)
	29:28	**Arbitration priority order between RCC and MSC** Default Value: 10b Access: R/W Arbitration priority order between RCC and MSC. 00b/11b: Invalid; default setting used. 10b: Default setting; RCC MSC (i.e., MSC has higher priority). 01b: RCC MSC (i.e., RCC has higher priority). Incf_csr_rcc_msc_pri[1:0].
	27:22	**Arbitration priority order between RCZ, STC, and HIZ** Default Value: 100100b Access: R/W Arbitration priority order between RCZ, STC, and HIZ. 100100b: Default setting; RCZ STC HIZ. (i.e., RCZ has lowest priority; HIZ has highest priority). 100001b: RCZ ; HIZ ; STC. 011000b: STC ; RCZ ; HIZ. 010010b: STC ; HIZ ; RCZ. 001001b: HIZ ; RCZ ; STC. 000110b: HIZ ; STC ; RCZ. Note: Others settings are invalid, and result in use of default. Incf_csr_rcz_stc_hiz_pri[5:0].

GARBCNTLREG - Arbiter Control Register

	21:19	**Write data port arbitration priority between Z client writes and L3$ evictions** Default Value: 010b Access: R/W Z Max Write Request Limit Count (GFXC_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (both Slice 0 and 1). Minimum count value must be = 1. Incf_csr_wdpagapz[2:0].
	18:16	**Write data port arbitration priority between C client writes and Z/L3$ writes/evictions** Default Value: 010b Access: R/W C Max Request Limit Count (GFXZ_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (both Slice 0 and 1). Minimum count value must be = 1. Incf_csr_wdpagapc[2:0].
	15	**Reserved** Access: RO
	14:12	**L3 Max Write Request Limit Count** Default Value: 100b Access: R/W L3 Max Write Request Limit Count (GFXL3_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1. Incf_csr_wdpagapl3[2:0].
	11:9	**Reserved** Access: RO
	8	**GAPs_fixarb_en** Default Value: 1b Access: R/W Incf_csr_gaps_fixarb_en.
	7	**GAPS TSV Credit fix Enable** Access: R/W Disables GAPS TSV fix for credit signal 0 (default): GAPS TSV fix for credit signal is disabled 1: GAPS TSV fix for credit signal is enabled This bit always needs to be programmed to 1 as part of the BIOS sequence Incf_csr_gaps_tsvfix_en.

GARBCNTLREG - Arbiter Control Register			
	6:0	**Reserved**	
		Access:	RO

Arbiter Mode Control Register

ARB_MODE - Arbiter Mode Control Register

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04030h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000000000000000b Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**Extra Register Bit 15** Default Value: 0b Access: R/W Bit 15 toggles (XOR) the meaning of Per Client Write Drop Enables (Register 40b4); If 0, drop per client happens as stated in register 40b4 definition; If 1, the meaning changes, and a 1 on a bit in register 40b4 means dont drop while 0 means drop. In this case, the default (for clients not included in 40b4) will be drop enabled.
	14	**Extra Register Bit 14** Default Value: 0b Access: R/W Reserved.
	13	**DC GDR** Default Value: 0b Access: R/W
	12	**HIZ GDR** Default Value: 0b Access: R/W
	11	**STC GDR** Default Value: 0b Access: R/W
	10	**BLB GDR** Default Value: 0b Access: R/W

ARB_MODE - Arbiter Mode Control Register

	9	**GAM PD GDR** Default Value: 0b Access: R/W
	8	**Extra Register Bit 8** Default Value: 0b Access: R/W **Description** Reserved.
	7:6	**Cacheability Attribute Override** Default Value: 00b Access: R/W 00b No override. 01b UC (LLC/eLLC) - Allocation age is don't care. 10b WT in LLC/eLLC - Aged is 3. 11b WB in LLC/eLLC - Aged is 3. The above conditions apply for the following conditions only: 1. Register overwrite except for GTT, CFG and L3 coherent wcil cycles 2. Read- GTTRD, CFGRD 3. Write- GTTWR, CFGWR, DMWR (with gam_ci_wcoherenttype[2:0]="001" WCIL* w/self snoop)
	5	**Extra Register Bit 5** Default Value: 0b Access: R/W Reserved.
	4	**VMC GDR Enable** Default Value: 0b Access: R/W When this bit is set, data requested from the VMC client is generated by the GDR Algorithm.
	3	**Texture Cache (MT) GDR Enable Bit** Default Value: 0b Access: R/W When this bit is set, data requested from the Texture Cache (MT) client is generated by the GDR algorithm.
	2	**Depth (RCZ) Cache GDR Enable bit** Default Value: 0b Access: R/W Depth Cache GDR enable bit. Project: All. Format: U1. When this bit is set, data requested from the Depth Cache client is generated by the GDR algorithm (See GDR algorithm in xxx section).

ARB_MODE - Arbiter Mode Control Register

	1	**Color Cache (RCC) GDR Enable Bit** Default Value: 0b Access: R/W When this bit is set, data requested from the Color Cache (RCC) client is generated by the GDR algorithm.
	0	**GTT Accesses GDR** Default Value: 0b Access: R/W When this bit is enabled along with the Client's GDR bit, PPGTT and GGTT requests for this memory access are also tagged as GDR to SQ.

ASL Storage

ASLS_0_2_0_PCI - ASL Storage

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	000FCh

This is a software scratch register. The exact bit register usage must be worked out in common between System BIOS and driver software.
For each device, the ASL control method requires two bits for DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for DGS (enable/disable requested), and two bits for DCS (enabled now/disabled now, connected or not).

DWord	Bit	Description
0	31:0	**Device Switching Storage** Default Value: 00000000000000000000000000000000b Access: R/W Software controlled usage to support device switching.

ATS Capability

ATS_CAP_0_2_0_PCI - ATS Capability

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000020
Size (in bits):	16
Address:	00204h

ATS Capability reports support for Device-TLBs on Device-2, compliant to PCI Express ATS specification.

DWord	Bit	Description
0	5	**Page Aligned Request** Default Value: 1b Access: RO Hardwired to 1, the Untranslated Address is always aligned to a 4096 byte boundary. Processor Graphics reports value of 1b indicating all VT-d and SVM translations are page-aligned.
	4:0	**Invalidate Queue Depth** Default Value: 00000b Access: RO The number of Invalidate Requests that the endpoint can accept before putting back pressure on the upstream connection. Hardwired to 0h, the function can accept 32 Invalidate Requests.

ATS Control

ATS_CTRL_0_2_0_PCI - ATS Control

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	16
Address:	00206h

DWord	Bit	Description
0	15	**ATS Enable** Default Value: 0b Access: R/W When Set, the function is enabled to cache translations. Processor graphics ignores this field, as GT uses GTLB as IOTLB and only pretends to software that it has a Device-TLB. Software is expected to Set this field before configuring extended context-entry for Device2 with Page Request Enable field Set. For compatibility, this field is implemeted as RW as software can read it to determine ATS enable status.
	14:5	**Reserved** Format: MBZ
	4:0	**Smallest Translation Unit** Default Value: 00000b Access: R/W This value indicates to the Endpoint the minimum number of 4096-byte blocks that is indicated in a Translation Completion or Invalidate Request. This is a power of 2 multiple and the number of blocks is 2^STU. A value of 0 indicates one block and value 1F indicates 2^31 blocks. For IGD this must be programmed to 0h for 4KB as smallest translation unit.

ATS Extended Capability Header

ATS_EXTCAP_0_2_0_PCI - ATS Extended Capability Header

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x3001000F
Size (in bits):	32
Address:	00200h

ATS Capability reports support for Device-TLBs on Device-2, compliant to PCI Express ATS specification.

DWord	Bit	Description
0	31:20	**Next Capability Offset** Default Value: 001100000000b Access: RO This is a hardwired pointer to the next item in the capabilities list. Value 300h in this field provides the offset for Page-Request Capability.
	19:16	**Version** Default Value: 0001b Access: RO Hardwired to capability version 1.
	15:0	**Capability ID** Default Value: 0000000000001111b Access: RO Hardwired to the ATS Extended Capability ID

AUD_CONFIG

AUD_CONFIG	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0070FA60
Access:	R/W
Size (in bits):	32
Address:	65000h-65003h
Name:	Audio Configuration Transcoder A
ShortName:	AUD_TCA_CONFIG
Power:	off/on
Reset:	soft
Address:	65100h-65103h
Name:	Audio Configuration Transcoder B
ShortName:	AUD_TCB_CONFIG
Power:	off/on
Reset:	soft
Address:	65200h-65203h
Name:	Audio Configuration Transcoder C
ShortName:	AUD_TCC_CONFIG
Power:	off/on
Reset:	soft

This register configures the audio output. There is one instance of this register per transcoder A/B/C. Each Transcoder is independent of the other.

DWord	Bit	Description
0	31:30	**Reserved**
	29	**N value Index** Value / Name / Description 0b / HDMI **[Default]** / N value read on bits 27:20 and 15:4 reflects HDMI N value. Bits 27:20 and 15:4 are programmable to any N value. Default h7FA6 when bit 28 is not set. 1b / DisplayPort / N value read on bits 27:20 and 15:4 reflects DisplayPort N value. Set this bit to 1 before programming N value register. When this bit is set to 1, 27:20 and 15:4 will reflect the current N value. Default is h8000 when bit 28 is not set.
	28	**N programming enable** This bit enables programming of N values for non-CEA modes. Please note that the transcoder to which audio is attached must be disabled when changing this field.

AUD_CONFIG

	27:20	**Upper N value** Default Value: 00000111b These are bits [19:12] of programmable N values for non-CEA modes. Bit 29 of this register must also be written in order to enable programming. Please note that the transcoder to which audio is attached must be disabled when changing this field. See bit 29 description for default values.
	19:16	**Pixel Clock HDMI** This is the target frequency of the CEA/HDMI video mode to which the audio stream is added. This value is used for generating N_CTS packets. This refers to only HDMI Pixel clock and does not refer to DisplayPort Link clock. DisplayPort Link clock does not require this programming. Note: The transcoder on which audio is attached must be disabled when changing this field.

Value	Name	Description
0b	**[Default]**	
0000b	25.2 / 1.001 MHz	25.2 / 1.001 MHz
0001b	25.2 MHz	25.2 MHz (Program this value for pixel clocks not listed in this field)
0010b	27 MHz	27 MHz
0011b	27 * 1.001 MHz	27 * 1.001 MHz
0100b	54 MHz	54 MHz
0101b	54 * 1.001 MHz	54 * 1.001 MHz
0110b	74.25 / 1.001 MHz	74.25 / 1.001 MHz
0111b	74.25 MHz	74.25 MHz
1000b	148.5 / 1.001 MHz	148.5 / 1.001 MHz
1001b	148.5 MHz	148.5 MHz
Others	Reserved	Reserved

	15:4	**Lower N value** Default Value: 111110100110b These are bits [11:0] of programmable N values for non-CEA modes. Bit 29 of this register must also be written in order to enable programming. Please note that the transcoder to which audio is attached must be disabled when changing this field. See bit 29 description for default values
	3	**Reserved**
	2:0	**Reserved**

AUD_DIP_ELD_CTRL_ST

AUD_DIP_ELD_CTRL_ST	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00005400
Access:	R/W
Size (in bits):	32
Address:	650B4h-650B7h
Name:	Audio Control State for DIP and ELD Transcoder A
ShortName:	AUD_TCA_DIP_ELD_CTRL_ST
Power:	off/on
Reset:	soft
Address:	651B4h-651B7h
Name:	Audio Control State for DIP and ELD Transcoder B
ShortName:	AUD_TCB_DIP_ELD_CTRL_ST
Power:	off/on
Reset:	soft
Address:	652B4h-652B7h
Name:	Audio Control State for DIP and ELD Transcoder C
ShortName:	AUD_TCC_DIP_ELD_CTRL_ST
Power:	off/on
Reset:	soft

There is one instance of this register per transcoder A/B/C.

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30:29	**DIP Port Select** Access: RO This read-only bit reflects which port is used to transmit the DIP data. This can only change when DIP is disabled. If one or more audio-related DIP packets is enabled and audio is enabled on a digital port, these bits will reflect the digital port to which audio is directed. For DP MST, this is the device select/pipe select.

Value	Name	Description
00b	Reserved **[Default]**	Reserved
01b	Digital Port B	Digital Port B
10b	Digital Port C	Digital Port C
11b	Digital Port D	Digital Port D

AUD_DIP_ELD_CTRL_ST

28:25 **Reserved**

Format:	MBZ

24:21 **DIP type enable status**

Access:	RO

These bits reflect the DIP types enabled. It can be updated while the port is enabled. Within 2 vblank periods, the DIP is guaranteed to have been transmitted. Disabling a DIP type results in setting the contents of that DIP buffer to zero. A reserved setting reflects a disabled DIP.

Value	Name	Description
0000b	**[Default]**	
XXX0b	DIP Disable	Audio DIP disabled
XXX1b	DIP Enable	Audio DIP enabled
XX0Xb	ACP Disable	Generic 1 (ACP) DIP disabled
XX1Xb	ACP Enable	Generic 1 (ACP) DIP enabled
X0XXb	Generic 2 Disable	Generic 2 DIP disabled
X1XXb	Generic 2 Enable	Generic 2 DIP enabled, can be used by ISRC1 or ISRC2
1XXXb	Reserved	Reserved

20:18 **DIP buffer index**

This field is used during read of different DIPs, and during read or write of ELD data. These bits are used as an index to their respective DIP or ELD buffers. When the index is not valid, the contents of the DIP will return all 0s.

Value	Name	Description
000b	Audio **[Default]**	Audio DIP (31 bytes of address space, 31 bytes of data)
001b	Gen 1	Generic 1 (ACP) Data Island Packet (31 bytes of address space, 31 bytes of data)
010b	Gen 2	Generic 2 (ISRC1) Data Island Packet (31 bytes of address space, 31 bytes of data)
011b	Gen 3	Generic 3 (ISRC2) Data Island Packet (31 bytes of address space, 31 bytes of data)
Others	Reserved	Reserved

17:16 **DIP transmission frequency**

Access:	RO

These bits reflect the frequency of DIP transmission for the DIP buffer type designated in bits 20:18. When writing DIP data, this value is also latched when the first DW of the DIP is written. When read, this value reflects the DIP transmission frequency for the DIP buffer designated in bits 20:18.

Value	Name	Description
00b	Disable **[Default]**	Disabled

AUD_DIP_ELD_CTRL_ST

		01b: Reserved: Reserved 10b: Send Once: Send Once 11b: Best Effort: Best effort (Send at least every other vsync)
	15	**Reserved** Format: MBZ
	14:10	**ELD buffer size** Default Value: 10101b Access: RO This field reflects the size of the ELD buffer in DWORDs (84 Bytes of ELD)
	9:5	**ELD access address** Selects the DWORD address for access to the ELD buffer (84 bytes). The value wraps back to zero when incremented past the max addressing value 0x1F. This field change takes effect immediately after being written. The read value indicates the current access address.
	4	**ELD ACK** Acknowledgement from the audio driver that ELD read has been completed
	3:0	**DIP access address** Selects the DWORD address for access to the DIP buffers. The value wraps back to zero when it incremented past the max addressing value of 0xF. This field change takes effect immediately after being written. The read value indicates the current access address.

AUD_EDID_DATA

AUD_EDID_DATA	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	65050h-65053h
Name:	Audio EDID Data Block Transcoder A
ShortName:	AUD_TCA_EDID_DATA
Power:	off/on
Reset:	soft
Address:	65150h-65153h
Name:	Audio EDID Data Block Transcoder B
ShortName:	AUD_TCB_EDID_DATA
Power:	off/on
Reset:	soft
Address:	65250h-65253h
Name:	Audio EDID Data Block Transcoder C
ShortName:	AUD_TCC_EDID_DATA
Power:	off/on
Reset:	soft

These registers contain the HDMI/DP data block from the EDID. The graphics driver reads the EDID and writes the structure to these registers. The vendor specific data block may be longer than 8 bytes, but the driver must not write more than 48 bytes to the buffer. The EDID format is Version 3 within the CEA-861B specification. The HDMI/DP Vendor Specific Data Block is described in version 1.1 of the HDMI specification.

These values are returned from the device as the HDMI/DP Vendor Specific Data Block response to a Get HDMI/DP Widget command.

Writing sequence:

- Video software sets ELD invalid, and sets the ELD access address to 0, or to the desired DWORD to be written.
- Video software writes ELD data 1 DWORD at a time. The ELD access address autoincrements with each DWORD write, wrapping around to address 0 when the max buffer address size of 0xF has been reached.
- Please note that software must write an entire DWORD at a time.
- Please note that the audio driver checks the valid bit with each byte read of the ELD. This means that the video driver can unilaterally write ELD irrespective of audio driver ELD read status.

Reading sequence:

AUD_EDID_DATA

- Video software sets the ELD access address to 0, or to the desired DWORD to be read.
- Video software reads ELD data 1 DWORD at a time. The ELD access address autoincrements with each DWORD read, wrapping around to address 0 when the max buffer address size of 0xF has been reached.

There is one instance of this register per transcoder A/B/C.

DWord	Bit	Description
0	31:0	**EDID Data Block** Please note that the contents of this buffer are not cleared when ELD is disabled. The contents of this buffer are cleared during FLR.

AUD_FREQ_CNTRL

AUD_FREQ_CNTRL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000010
Access:	R/W
Size (in bits):	32
Address:	65900h-65903h
Name:	Audio BCLK Frequency Control
ShortName:	AUD_FREQ_CNTRL
Power:	off/on
Reset:	soft

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15	**T-Mode** Indicates whether SDI is operating in 1T mode or 2T mode. BIOS or System Software must pre-program the T-mode register. a. before the iDISPLAY Audio Link is brought out from Link Reset, b. to a value which is consistent with the value of the its counterpart T-mode bit in the Audio Controller (inside the Skylake PCH). c. to a value which is within the electrical capabilities of the platform. Note that 2T mode is prohibited from being used with any BCLK frequency which has an odd number of bit cells. Example, 2T mode is incompatible with BCLK=6MHz (125 bit cells). Value / Name / Description: 0b / Enable **[Default]** / 2T mode with sdi data held for 2 bit clks. 1b / Disable / 1T Mode with sdi data held for 1 bit clock only.
	14:5	**Reserved** Format: MBZ
	4	**96MHz BCLK** Default Value: 1b Indicates that iDISPLAY Audio Link will run at 96MHz. This bit is defaulted to 1. BIOS or System Software must pre-program B96 before the iDISPLAY Audio Link is brought out from reset.
	3	**48MHz BCLK** Default Value: 0b Indicates that iDISPLAY Audio Link will run at 48MHz. This bit is defaulted to 0. BIOS or System Software must pre-program B96 before the iDISPLAY Audio Link is brought out from reset.

AUD_FREQ_CNTRL		
	2:0	**Reserved** Format: MBZ

AUD_INFOFR

AUD_INFOFR	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	65054h-65057h
Name:	Audio Widget Data Island Packet Transcoder A
ShortName:	AUD_TCA_INFOFR
Power:	off/on
Reset:	soft
Address:	65154h-65157h
Name:	Audio Widget Data Island Packet Transcoder B
ShortName:	AUD_TCB_INFOFR
Power:	off/on
Reset:	soft
Address:	65254h-65257h
Name:	Audio Widget Data Island Packet Transcoder C
ShortName:	AUD_TCC_INFOFR
Power:	off/on
Reset:	soft

When the IF type or dword index is not valid, the contents of the DIP will return all 0s.
These values are programmed by the audio driver in an HDMI/DP Widget Set command. They are returned one byte at a time from the device on the HD audio bus as the HDMI/DP DIP response to a Get HDMI/DP Widget command. To fetch a specific byte, the audio driver should send an HDMI/DP Widget HDMI/DP DIP Index Pointer Set command to set the index, then fetch the indexed byte using the HDMI/DP DIP get.

DWord	Bit	Description
0	31:0	**Data Island Packet Data** This reflects the contents of the DIP indexed by the DIP access address. The contents of this buffer are cleared during function reset or HD audio link reset.

AUD_M_CTS_ENABLE

AUD_M_CTS_ENABLE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	65028h-6502Bh
Name:	Audio M and CTS Programming Enable Transcoder A
ShortName:	AUD_TCA_M_CTS_ENABLE
Power:	off/on
Reset:	soft
Address:	65128h-6512Bh
Name:	Audio M and CTS Programming Enable Transcoder B
ShortName:	AUD_TCB_M_CTS_ENABLE
Power:	off/on
Reset:	soft
Address:	65228h-6522Bh
Name:	Audio M and CTS Programming Enable Transcoder C
ShortName:	AUD_TCC_M_CTS_ENABLE
Power:	off/on
Reset:	soft

There is one instance of this register per transcoder A/B/C.

DWord	Bit	Description
0	31:22	**Reserved**
	21	**CTS M value Index** Value / Name / Description 0b / CTS **[Default]** / CTS value read on bits 19:0 reflects CTS value. Bit 19:0 is programmable to any CTS value. default is 0 1b / M / M value read on bits 19:0 reflects DisplayPort M value. Set this bit to 1 before programming M value register. When this is set to 1 19:0 will reflect the current M value
	20	**Enable CTS or M prog** When set will enable CTS or M programming.

AUD_M_CTS_ENABLE		
	19:0	**CTS programming** These are bits [19:0] of programmable CTS values for non-CEA modes. Bit 21 of this register must also be written in order to enable programming. Please note that the transcoder to which audio is attached must be disabled when changing this field.

AUD_MISC_CTRL

AUD_MISC_CTRL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000044
Access:	R/W
Size (in bits):	32
Address:	65010h-65013h
Name:	Audio Converter 1 Misc Control
ShortName:	AUD_C1_MISC_CTRL
Power:	off/on
Reset:	soft
Address:	65110h-65113h
Name:	Audio Converter 2 Misc Control
ShortName:	AUD_C2_MISC_CTRL
Power:	off/on
Reset:	soft
Address:	65210h-65213h
Name:	Audio Converter 3 Misc Control
ShortName:	AUD_C3_MISC_CTRL
Power:	off/on
Reset:	soft
There is one instance of this register per audio converter 1/2/3.	

DWord	Bit	Description
0	31:9	**Reserved** Format: MBZ
	8	**Reserved**
	7:4	**Output Delay** Default Value: 0100b The number of samples between when the sample is received from the HD Audio link and when it appears as an analog signal at the pin.
	3	**Reserved** Format: MBZ
	2	**Sample Fabrication EN bit** Access: R/W

AUD_MISC_CTRL

This bit indicates whether internal fabrication of audio samples is enabled during a link underrun.

Value	Name	Description
0b	Disable	Audio fabrication disabled
1b	Enable **[Default]**	Audio fabrication enabled

1 **Pro Allowed**

Access:	R/W

By default, the audio device is configured to consumer mode and does not allow the mode to be changed to professional mode by an HD Audio verb. When Pro is allowed by setting this configuration bit, the HD Audio codec allows a verb to set the device into professional mode.
Note: Setting this configuration bit does not change the default Pro bit value to be 1. Pro must be set to 1 through the normal process, using a verb.

Value	Name	Description
0b	Consumer **[Default]**	Consumer use only
1b	Professional	Professional use allowed

0 **Reserved**

Format:	MBZ

AUD_PIN_ELD_CP_VLD

AUD_PIN_ELD_CP_VLD

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	650C0h-650C3h
Name:	Audio Pin ELD and CP Ready Status
ShortName:	AUD_PIN_ELD_CP_VLD
Power:	off/on
Reset:	soft

DWord	Bit	Description
0	31:12	**Reserved**
	11	**Audio InactiveC** Inactive: When this bit is set, a digital display sink device has been attached but not active for streaming audio. Value / Name / Description: 0b / Disable / Device is active for streaming audio data 1b / Enable / Device is connected but not active
	10	**Audio Output EnableC** This bit directs audio to the device connected to this transcoder. When enabled along with Inactive set to 0 and audio data is available, the audio data will be combined with the video data and sent over this transcoder. The audio unit uses the status of this bit to indicate presence of the HDMI/DP output to the audio driver. Value / Name / Description: 0b / Disable / No Audio output 1b / Valid / Audio is enabled
	9	**CP ReadyC** This R/W bit reflects the state of CP request from the audio unit. When an audio CP request has been serviced, it must be reset to 1 by the video software to indicate that the CP request has been serviced. This is transcoder based. Software should add a delay of 1ms before updating the CP ready bit. This is needed to make sure that all the pending unsolicited responses are cleared (transmitted to HD audio) before CP ready unsolicited responses is generated. This is needed in case of DP MST is enabled and when many changes to PD, ELDV and CP ready bits are done during mode set. Value / Name / Description: 0b / Pending or Not Ready / CP request pending or not ready to receive requests 1b / Ready / CP request ready

AUD_PIN_ELD_CP_VLD

8	**ELD validC** This R/W bit reflects the state of the ELD data written to the ELD RAM. After writing the ELD data, the video software must set this bit to 1 to indicate that the ELD data is valid. At audio codec initialization, or on a hotplug event, this bit is set to 0 by the video software. This bit is reflected in the audio pin complex widget as the ELD valid status bit. This is transcoder based.

Value	Name	Description
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)
1b	Valid	ELD data valid (Set by video software only)

7	**Audio InactiveB** Inactive: When this bit is set, a digital display sink device has been attached but not active for streaming audio.

Value	Name	Description
0b	Disable	Device is active for streaming audio data
1b	Enable	Device is connected but not active

6	**Audio Output EnableB** This bit directs audio to the device connected to this transcoder. When enabled along with Inactive set to 0 and audio data is available, the audio data will be combined with the video data and sent over this transcoder. The audio unit uses the status of this bit to indicate presence of the HDMI/DP output to the audio driver. This is transcoder based.

Value	Name	Description
0b	Disable	No audio output
1b	Enable	Audio is enabled

5	**CP ReadyB** See CP_ReadyC description.

Value	Name	Description
0b	Not Ready	CP request pending or not ready to receive requests
1b	Ready	CP request ready

4	**ELD validB** See ELD_validC descripion.

Value	Name	Description
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)
1b	Valid	ELD data valid (Set by video software only)

3	**Audio InactiveA** Inactive: When this bit is set, a digital display sink device has been attached but not active for streaming audio.

Value	Name	Description
0b	Disable	Device is active for streaming audio data
1b	Enable	Device is connected but not active

AUD_PIN_ELD_CP_VLD

	2	**Audio Output EnableA** This bit directs audio to the device connected to this transcoder. When enabled along with Inactive set to 0 and audio data is available, the audio data will be combined with the video data and sent over this transcoder. The audio unit uses the status of this bit to indicate presence of the HDMI/DP output to the audio driver. This is transcoder based.

Value	Name	Description
0b	Disable	No audio output
1b	Enable	Audio is enabled

	1	**CP ReadyA** See CP_ReadyC description.

Value	Name	Description
0b	Not Ready	CP request pending or not ready to receive requests
1b	Ready	CP request ready

	0	**ELD validA** See ELD_validC descripion.

Value	Name	Description
0b	Invalid	ELD data invalid (default, when writing ELD data, set 0 by software)
1b	Valid	ELD data valid (Set by video software only)

AUD_PIN_PIPE_CONN_ENTRY_LNGTH

AUD_PIN_PIPE_CONN_ENTRY_LNGTH

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000001
Access:	RO
Size (in bits):	32

Address:	650A8h-650ABh
Name:	Audio Connection List Entry and Length Transcoder A
ShortName:	AUD_TCA_PIN_PIPE_CONN_ENTRY_LNGTH_RO
Power:	off/on
Reset:	soft

Address:	651A8h-651ABh
Name:	Audio Connection List Entry and Length Transcoder B
ShortName:	AUD_TCB_PIN_PIPE_CONN_ENTRY_LNGTH_RO
Power:	off/on
Reset:	soft

Address:	652A8h-652ABh
Name:	Audio Connection List Entry and Length Transcoder C
ShortName:	AUD_TCC_PIN_PIPE_CONN_ENTRY_LNGTH_RO
Power:	off/on
Reset:	soft

These values are returned from the device as the Connection List Length response to a Get Pin Widget command or Get Device Widget command if DP MST. There is one instance of this register per transcoder A/B/C.

DWord	Bit	Description
0	31:16	**Reserved**
	15:8	**Connection List Entry** Connection to Convertor Widget Node 0x03
	7	**Long Form** This bit indicates whether the items in the connection list are long form or short form. This bit is hardwired to 0 (items in connection list are short form)
	6:0	**Connection List Length** Default Value: 0000001b This field indicates the number of items in the connection list. If this field is 2, there is only one hardwired input possible, which is read from the Connection List, and there is no Connection Select Control.

AUD_PIPE_CONN_SEL_CTRL

AUD_PIPE_CONN_SEL_CTRL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00030303
Access:	RO
Size (in bits):	32
Address:	650ACh-650AFh
Name:	Audio Pipe Connection Select Control
ShortName:	AUD_PIN_PIPE_CONN_SEL_CTRL_RO
Power:	off/on
Reset:	soft

These values are returned from the device as the Connection List Length response to a Get Pin Widget command or Get Device Widget command for DP MST.

DWord	Bit	Description
0	31:24	**Reserved**
	23:16	**Connection select Control D** Connection Index Currently Set [Default 0x00], Port D Widget is set to 0x02 Value: 03h — Name: **[Default]**
	15:8	**Connection select Control C** Connection Index Currently Set [Default 0x00], Port C Widget is set to 0x01 Value: 03h — Name: **[Default]**
	7:0	**Connection select Control B** Connection Index Currently Set [Default 0x00], Port B Widget is set to 0x00 Value: 03h — Name: **[Default]**

AUD_PWRST

AUD_PWRST

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0FFFFFFF
Access:	RO
Size (in bits):	32
Address:	6504Ch-6504Fh
Name:	Audio Power State Read Only
ShortName:	AUD_PWRST_RO
Power:	off/on
Reset:	soft

These values are returned from the device as the Power State response to a Get Audio Function Group command.

DWord	Bit	Description
0	31:28	**Reserved**
	27:26	**Func Grp Dev PwrSt Curr** Format: **Audio Power State Format** Function Group Device current power state Value: 11b Name:
	25:24	**Func Grp Dev PwrSt Set** Format: **Audio Power State Format** Function Group Device power state that was set Value: 11b Name:
	23:22	**Converter3 Widget PwrSt Curr** Format: **Audio Power State Format** Converter3 Widget current power state Value: 11b Name:
	21:20	**Converter3 Widget PwrSt Req** Format: **Audio Power State Format** Converter3 Widget power state that was requested by audio software Value: 11b Name:

AUD_PWRST

	19:18	**Convertor2 Widget PwrSt Curr** Format: **Audio Power State Format** Converor2 Widget current power state **Value** / **Name** 11b
	17:16	**Convertor2 Widget PwrSt Req** Format: **Audio Power State Format** Converter2 Widget power state that was requested by audio software **Value** / **Name** 11b
	15:14	**Convertor1 Widget PwrSt Curr** Format: **Audio Power State Format** Converter1 Widget current power state **Value** / **Name** 11b
	13:12	**Convertor1 Widget PwrSt Req** Format: **Audio Power State Format** Converter1 Widget power state that was requested by audio software **Value** / **Name** 11b
	11:10	**PinD Widget PwrSt Curr** Format: **Audio Power State Format** PinD Widget current power stateFor DP MST this represents Device3 power state **Value** / **Name** 11b
	9:8	**PinD Widget PwrSt Set** Format: **Audio Power State Format** PinD Widget power state that was setFor DP MST this represents Device3 power state **Value** / **Name** 11b
	7:6	**PinC Widget PwrSt Curr** Format: **Audio Power State Format** PinC Widget current power stateFor DP MST this represents Device2 power state **Value** / **Name** 11b

AUD_PWRST

	5:4	**PinC Widget PwrSt Set** Format: **Audio Power State Format** PinC Widget power state that was setFor DP MST this represents Device2 power state **Value** / **Name** 11b
	3:2	**PinB Widget PwrSt Curr** Format: **Audio Power State Format** PinB Widget current power stateFor DP MST this represents Device1 power state **Value** / **Name** 11b
	1:0	**PinB Widget PwrSt Set** Format: **Audio Power State Format** PinB Widget power state that was setFor DP MST this represents Device1 power state **Value** / **Name** 11b

AUD_RID

AUD_RID	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Access:	RO
Size (in bits):	32
Address:	65024h-65027h
Name:	Audio Revision ID Read Only
ShortName:	AUD_RID_RO
Power:	off/on
Reset:	soft

These values are returned from the device as the Revision ID response to a Get Root Node command.

DWord	Bit	Description
0	31:24	**Reserved**
	23:20	**Major Revision** Default Value: 1h The major revision number (left of the decimal) of the HD Audio Spec to which the codec is fully compliant. This field is hardwired within the device.
	19:16	**Minor Revision** Default Value: 0h The minor revision number (rights of the decimal) or dot number of the HD Audio Spec to which the codec is fully compliant. This field is hardwired within the device.
	15:8	**Revision ID** Default Value: 00h The vendor revision number for this given Device ID. This field is hardwired within the device.
	7:0	**Stepping ID** Default Value: 00h An optional vendor stepping number within the given Revision ID. This field is hardwired within the device.

AUD_VID_DID

AUD_VID_DID

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x8086280B
Access:	RO
Size (in bits):	32
Address:	65020h-65023h
Name:	Audio Vendor ID / Device ID Read Only
ShortName:	AUD_VID_DID_RO
Power:	off/on
Reset:	soft

These values are returned from the device as the Vendor ID/ Device ID response to a Get Root Node command.

DWord	Bit	Description
0	31:16	**Vendor ID** Default Value: 8086h Used to identify the codec within the PnP system. This field is hardwired within the device.
	15:0	**Device ID** Constant used to identify the codec within the PnP system. This field is set by the device hardware. Value: 280Bh — Name: **[Default]**

AUD_WNIC_PCR_CNTRL

AUD_WNIC_PCR_CNTRL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000001
Access:	R/W
Size (in bits):	32
Address:	659A8h-659ABh
Name:	Audio WNIC PCR Control
ShortName:	AUD_WNIC_PCR_CNTRL
Power:	off/on
Reset:	soft

DWord	Bit	Description
0	31	**PCRupdate_ENABLE** Enable to send the PCR updates to the WNIC. Value / Name / Description: 1b / Enable / When set to 1, the Audio engine will capture the PCR value from Display counter depending on the programmed select value below and send the captured PCR value to the WNIC as a message. 0b / Disable **[Default]** / When set to 0, PCR updates are not sent to the WNIC.
	30:3	**Reserved** Format: MBZ
	2:0	**PCR counter bit select** Value / Name / Description: 001b / 4.8ms **[Default]** / When set the PCR value is capture whenever the bit 17 of the counter changes. 010b / 9.6ms / When set the PCR value is capture whenever the bit 18 of the counter changes. 100b / 19.2ms / When set the PCR value is capture whenever the bit 19 of the counter changes. Others / Reserved / Cannot have two or more bits set at the same time. If set then behavior is unpredictable.

Audio Codec Interrupt Definition

Audio Codec Interrupt Definition

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44480h-4448Fh
Name:	Audio Codec Interrupts
ShortName:	AUD_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the Audio Codec Interrupt registers.
0x44480 = ISR
0x44484 = IMR
0x44488 = IIR
0x4448C = IER

DWord	Bit	Description
0	31	**Audio_Power_State_change_DDI_D** The ISR is an active high pulse when there is a power state change for audio for DDI D.
	30	**Audio_Power_State_change_DDI_C** The ISR is an active high pulse when there is a power state change for audio for DDI C.
	29	**Audio_Power_State_change_DDI_B** The ISR is an active high pulse when there is a power state change for audio for DDI B.
	28	**Audio_Power_State_change_WD_0** Description The ISR is an active high pulse when there is a power state change for audio for WD 0.
	27	**Spare 27**
	26	**Spare 26**
	25	**Spare 25**
	24	**Spare 24**
	23	**Spare 23**
	22	**Spare 22**
	21	**Spare 21**
	20	**Spare 20**
	19	**Spare 19**
	18	**Spare 18**
	17	**Spare 17**

Audio Codec Interrupt Definition		
	16	**Spare 16**
	15	**Spare 15**
	14	**Reserved**
	13	**Reserved**
	12	**Spare 12**
	11	**Spare 11**
	10	**Reserved**
	9	**Reserved**
	8:7	**Unused_Int_8_7** These interrupts are currently unused.
	6	**Reserved**
	5	**Reserved**
	4:3	**Unused_Int_4_3** These interrupts are currently unused.
	2	**Reserved**
	1	**Reserved**
	0	**Spare 0**

Auto Draw End Offset

3DPRIM_END_OFFSET - Auto Draw End Offset	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02420h-02423h

DWord	Bit	Description
0	31:0	**End Offset** Format: U32 This register is used to store the end offset value used by the Vertex Fetch to determine when to stop processing the 3D_PRIMITIVE command. This register is valid when the End Offset Enable is set in the 3D_PRIMITIVE command.

AVC GAM Slave Counter High part

AVC_GAM_SLAVE_CTR_H - AVC GAM Slave Counter High part	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	048A0h
Name:	AVC slave counter high for VDBOX0
ShortName:	AVC_VDBOX0_CTR_H
Address:	048A4h
Name:	AVC slave counter high for VDBOX1
ShortName:	AVC_VDBOX1_CTR_H
Address:	048A8h
Name:	AVC slave counter high for VDBOX2
ShortName:	AVC_VDBOX2_CTR_H
Address:	048ACh
Name:	AVC slave counter high for VDBOX3
ShortName:	AVC_VDBOX3_CTR_H
Address:	048B0h
Name:	AVC slave counter high for VDBOX4
ShortName:	AVC_VDBOX4_CTR_H
Address:	048B4h
Name:	AVC slave counter high for VDBOX5
ShortName:	AVC_VDBOX5_CTR_H
Address:	048B8h
Name:	AVC slave counter high for VDBOX6
ShortName:	AVC_VDBOX6_CTR_H
Address:	048BCh
Name:	AVC slave counter high for VDBOX7
ShortName:	AVC_VDBOX7_CTR_H

DWord	Bit	Description
0	31:0	**AVC GAM SLave Counter High** Default Value: 00000000h Access: R/W AVC GAM Slave counter[63:32]

AVC GAM Slave Counter Low part

AVC_GAM_SLAVE_CTR_L - AVC GAM Slave Counter Low part

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32

Address:	04860h
Name:	AVC slave counter low for VDBOX0
ShortName:	AVC_VDBOX0_CTR_L

Address:	04864h
Name:	AVC slave counter low for VDBOX1
ShortName:	AVC_VDBOX1_CTR_L

Address:	04868h
Name:	AVC slave counter low for VDBOX2
ShortName:	AVC_VDBOX2_CTR_L

Address:	0486Ch
Name:	AVC slave counter low for VDBOX3
ShortName:	AVC_VDBOX3_CTR_L

Address:	04870h
Name:	AVC slave counter low for VDBOX4
ShortName:	AVC_VDBOX4_CTR_L

Address:	04874h
Name:	AVC slave counter low for VDBOX5
ShortName:	AVC_VDBOX5_CTR_L

Address:	04878h
Name:	AVC slave counter low for VDBOX6
ShortName:	AVC_VDBOX6_CTR_L

Address:	0487Ch
Name:	AVC slave counter low for VDBOX7
ShortName:	AVC_VDBOX7_CTR_L

DWord	Bit	Description
0	31:0	**AVC GAM SLave Counter Low** Default Value: 00000000h Access: R/W AVC GAM Slave counter[31:0]

Base Data of Stolen Memory

BDSM_0_0_0_PCI - Base Data of Stolen Memory

Register Space:	PCI: 0/0/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	000B0h

This register contains the base address of graphics data stolen DRAM memory.

DWord	Bit	Description
0	31:20	**Graphics Base of Stolen Memory** Default Value: 000000000000b Access: R/W Lock This register contains bits 31 to 20 of the base address of stolen DRAM memory. BIOS determines the base of graphics stolen memory by subtracting the graphics stolen memory size (PCI Device 0 offset 52 bits 6:4) from TOLUD (PCI Device 0 offset BC bits 31:20).
	19:1	**Reserved** Format: MBZ
	0	**Lock** Default Value: 0b Access: R/W Key Lock This bit will lock all writeable settings in this register, including itself.

Base of GTT Stolen Memory

BGSM_0_0_0_PCI - Base of GTT Stolen Memory

Register Space:	PCI: 0/0/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	000B4h

This register contains the base address of stolen DRAM memory for the GTT.

DWord	Bit	Description
0	31:20	**Graphics Base of GTT Stolen Memory** Default Value: 000000000001b Access: R/W Lock This register contains the base address of stolen DRAM memory for the GTT. BIOS determines the base of GTT stolen memory by subtracting the GTT graphics stolen memory size (PCI Device 0 offset 52 bits 11:8) from the Graphics Base of Data Stolen Memory (PCI Device 0 offset B0 bits 31:20).
	19:1	**Reserved** Format: MBZ
	0	**Lock** Default Value: 0b Access: R/W Key Lock This bit will lock all writeable settings in this register, including itself.

Batch Address Difference Register

BB_ADDR_DIFF - Batch Address Difference Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02154h-02157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_RCSUNIT

Address:	12154h-12157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VCSUNIT0

Address:	1A154h-1A157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VECSUNIT

Address:	1C154h-1C157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VCSUNIT1

Address:	22154h-22157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_BCSUNIT

This register contains the difference between the start of the last batch and where the last initiated Batch Buffer is currently fetching commands.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Address Difference** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned difference between the starting address of the batch buffer and where the last initiated Batch Buffer is currently fetching commands.
	1:0	**Reserved** Format: MBZ

Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02148h-0214Bh
Name:	Batch Buffer Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_RCSUNIT

Address:	12148h-1214Bh
Name:	Batch Buffer Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_VCSUNIT0

Address:	1A148h-1A14Bh
Name:	Batch Buffer Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_VECSUNIT

Address:	1C148h-1C14Bh
Name:	Batch Buffer Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_VCSUNIT1

Address:	22148h-2214Bh
Name:	Batch Buffer Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_BCSUNIT

Description

This register gets updated with the DWord-aligned graphics memory address of the PREEMPTABLE command in the batch buffer on which preemption has occurred.
This register gets updated with the DWord-aligned graphics memory address of the command following the MI_BATCH_START corresponding to the second level batch buffer, when the preemption has occurred in the second level batch buffer.
This register value should be looked at only when the preemption has occurred in the batch buffer. This is indicated by "Ring/Batch Indicator" in "RING_BUFFER_HEAD_PREEMPT_REG". This register value retains its previous value and doesn't change when the preemption occurs on a preemptable command in ring buffer.
Preemption is triggered by valid UHPTR in ring buffer mode of scheduling and by a pending execlist in Exec-List mode of scheduling.
This is a global register and context save/restored as part of power context image.

Preemptable Commands	Source

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

MI_ARB_CHECK 3D_PRIMITIVE GPGPU_WALKER MEDIA_STATE_FLUSH PIPE_CONTROL (Only in GPGPU mode of pipeline selection) MI_ATOMIC (Post Sync Operation set in GPGPU mode of pipeline selection) MI_SEMAPHORE_SIGNAL (Post Sync Operation set in GPGPU mode of pipeline selection)	RenderCS

Preemptable Commands	Source
MI_ARB_CHECK	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Head Pointer** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address of the PREEMPTABLE command in a batch buffer where the Preemption has occurred.
	1:0	**Reserved** Format: MBZ

Batch Buffer Head Pointer Register

BB_ADDR - Batch Buffer Head Pointer Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32

Address:	02140h-02143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_RCSUNIT

Address:	12140h-12143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VCSUNIT0

Address:	1A140h-1A143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VECSUNIT

Address:	1C140h-1C143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VCSUNIT1

Address:	22140h-22143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_BCSUNIT

Description

This field specifies the DWord-aligned Graphics Memory Address of commands being fetched from the first level batch buffer. This register have valid values only when the “Valid” bit is set to‘0’.

Programming Notes

Programming Restriction: This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Head Pointer** Format: GraphicsAddress[31:2] **Description** This field specifies the DWord-aligned Graphics Memory Address of commands being fetched from the first level batch buffer. “Valid” bit will be ‘0’ when there is no active batch buffer and this field has no significance.

BB_ADDR - Batch Buffer Head Pointer Register

		This field specifies the DWord-aligned Graphics Memory Address of commands being fetched for the most recently initiated batch buffer. This register have valid values only when the "Valid" bit is set to'0'. Level of the batch buffer is indicated based on the Batch Buffer Stack Pointer value in BB_STATE register. • Stack Pointer holding a value '0' indicates First Level batch buffer. • Stack Pointer holding a value '1' indicates Second Level batch buffer. • Stack Pointer holding a value '2' indicates Third Level batch buffer.
	1	**Reserved** Format: MBZ
	0	**Valid** Format: U1

Value	Name	Description
0h	Invalid **[Default]**	Batch buffer Invalid
1h	Valid	Batch buffer Valid

Batch Buffer Per Context Pointer

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	021C0h-021C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_RCSUNIT
Address:	121C0h-121C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VCSUNIT0
Address:	1A1C0h-1A1C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VECSUNIT
Address:	1C1C0h-1C1C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VCSUNIT1
Address:	221C0h-221C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_BCSUNIT

This register is used to program the batch buffer address to be executed between context restore and execution of ring/execution list if enabled. This will only get executed due to regular context save/restore and not during power restore. This register is part of the execution list context and will be executed per context. Only supported if execution list is enabled. There is no preempting workloads within the Per Context Batch Buffer.

Programming Notes	Source
BlitterCS/VideoCS/VideoCS2/VideoEnhancementCS: This register functionality is not supported and must not be programmed for these command streamers.	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Render CS: Per Context Batch Buffer execution must not look at the MI_RS_CONTROL or Wait For Event status that are restored for the corresponding context. Ex: A context with MI_RS_CONTROL status with RS disabled doesn't stop Render CS from triggering Resource Streamer to execute Per Context Batch Buffer when "**RS Enabled Batch Buffer Per Context**" is set.	RenderCS

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

RenderCS: The following commands are not supported within a Per Context Batch Buffer: **Command Name** MI_WAIT_FOR_EVENT MI_ARB_CHECK MI_RS_CONTROL MI_REPORT_HEAD MI_URB_ATOMIC_ALLOC MI_SUSPEND_FLUSH MI_TOPOLOGY_FILTER MI_RS_CONTEXT MI_SET_CONTEXT MI_URB_CLEAR MI_SEMAPHORE_WAIT (Memory Poll Mode).Note: MI_SEMAPHORE_WAIT in register poll mode is supported. MI_SEMAPHORE_SIGNAL MI_BATCH_BUFFER_START MI_CONDITIONAL_BATCH_BUFFER_END MEDIA_OBJECT_WALKER GPGPU_WALKER 3DPRIMITIVE 3DSTATE_BINDING_TABLE_POINTERS_VS 3DSTATE_BINDING_TABLE_POINTERS_HS 3DSTATE_BINDING_TABLE_POINTERS_DS 3DSTATE_BINDING_TABLE_POINTERS_GS 3DSTATE_BINDING_TABLE_POINTERS_PS 3DSTATE_GATHER_CONSTANT_VS 3DSTATE_GATHER_CONSTANT_GS 3DSTATE_GATHER_CONSTANT_HS 3DSTATE_GATHER_CONSTANT_DS 3DSTATE_GATHER_CONSTANT_PS 3DSTATE_DX9_CONSTANTF_VS 3DSTATE_DX9_CONSTANTF_HS 3DSTATE_DX9_CONSTANTF_DS 3DSTATE_DX9_CONSTANTF_GS 3DSTATE_DX9_CONSTANTF_PS	RenderCS

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

3DSTATE_DX9_CONSTANTI_VS
3DSTATE_DX9_CONSTANTI_HS
3DSTATE_DX9_CONSTANTI_DS
3DSTATE_DX9_CONSTANTI_GS
3DSTATE_DX9_CONSTANTI_PS
3DSTATE_DX9_CONSTANTB_VS
3DSTATE_DX9_CONSTANTB_HS
3DSTATE_DX9_CONSTANTB_DS
3DSTATE_DX9_CONSTANTB_GS
3DSTATE_DX9_CONSTANTB_PS
3DSTATE_DX9_LOCAL_VALID_VS
3DSTATE_DX9_LOCAL_VALID_DS
3DSTATE_DX9_LOCAL_VALID_HS
3DSTATE_DX9_LOCAL_VALID_GS
3DSTATE_DX9_LOCAL_VALID_PS
3DSTATE_DX9_GENERATE_ACTIVE_VS
3DSTATE_DX9_GENERATE_ACTIVE_HS
3DSTATE_DX9_GENERATE_ACTIVE_DS
3DSTATE_DX9_GENERATE_ACTIVE_GS
3DSTATE_DX9_GENERATE_ACTIVE_PS
3DSTATE_BINDING_TABLE_EDIT_VS
3DSTATE_BINDING_TABLE_EDIT_GS
3DSTATE_BINDING_TABLE_EDIT_HS
3DSTATE_BINDING_TABLE_EDIT_DS
3DSTATE_BINDING_TABLE_EDIT_PS
3DSTATE_CONSTANT_VS
3DSTATE_CONSTANT_GS
3DSTATE_CONSTANT_PS
3DSTATE_CONSTANT_HS
3DSTATE_CONSTANT_DS
PIPECONTROL

DWord	Bit	Description
0	31:12	**Batch Buffer Per Context Address** Format: U20 Pointer to the Context in memory to be executed as a batch.

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

	11:3	**Reserved** Format: MBZ
	2	**Reserved**
	1	**RS Enabled Batch Buffer Per Context** Format: U1 If set, the command stream will enable the RS to parse commands. **Programming Notes** This must be set when programming the resource streamer pool commands (3DSTATE_BINDING_TABLE_POOL_ALLOC, 3DSTATE_GATHER_POOL_ALLOC, and 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC) in order for the pool alloc fields to be valid in both the render engine and resource streamer.
	0	**Batch Buffer Per Context Valid** Format: U1 If set, the command stream will execute the context from the **Batch Buffer Per Context Address** prior to the execution of actual submitted workloads.

Batch Buffer Start Head Pointer Register

BB_START_ADDR - Batch Buffer Start Head Pointer Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02150h-02153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_RCSUNIT
Address:	12150h-12153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VCSUNIT0
Address:	1A150h-1A153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VECSUNIT
Address:	1C150h-1C153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VCSUNIT1
Address:	22150h-22153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_BCSUNIT

This register contains the address specified in the last MI_START_BATCH_BUFFER command.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Start Head Pointer** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer starting address.
	1:0	**Reserved** Format: MBZ

Batch Buffer Start Upper Head Pointer Register

BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02170h-02173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_RCSUNIT

Address:	12170h-12173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VCSUNIT0

Address:	1A170h-1A173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VECSUNIT

Address:	1C170h-1C173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VCSUNIT1

Address:	22170h-22173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_BCSUNIT

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space specified in the last MI_START_BATCH_BUFFER command.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Head Pointer Upper DWORD** Format: GraphicsAddress[47:32]

Batch Buffer State Register

BB_STATE - Batch Buffer State Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	02110h-02113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_RCSUNIT
Address:	12110h-12113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VCSUNIT0
Address:	1A110h-1A113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VECSUNIT
Address:	1C110h-1C113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VCSUNIT1
Address:	22110h-22113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_BCSUNIT

Description

This register contains the attributes of the current batch buffer initiated from the Ring Buffer.

This register should not be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer.
This register is saved and restored with context.

Programming Notes

Contents of this register are valid only when "Valid" bit in BB_ADDR register is set.

DWord	Bit	Description
0	31:10	**Reserved** Format: MBZ
	9:8	**Reserved**

BB_STATE - Batch Buffer State Register

	7	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	7	**Resource Streamer Enable** Source: RenderCS Format: U1 When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer.
	6	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	6	**Clear Command Buffer Enable** Source: RenderCS Format: U1 If set the batch buffer is getting executed from the Write Once protected memory area. The address of the batch buffer is an offset into the WOPCM area.
	5	**Address Space Indicator** Note: This field reflects the effective address space indicator security level and may not be the same as the Address Space Indicator written using MI_BATCH_BUFFER_START.

Value	Name	Description
0h	GGTT **[Default]**	This Batch buffer is located in GGTT memory and is privileged
1h	PPGTT	This Batch buffer is located in PPGTT memory and is non-privileged.

	4	**Reserved**
	4	**Reserved** Source: BlitterCS Exists If: //BCS Format: MBZ
	3:2	**Reserved** Format: MBZ
	1:0	**Reserved**

Batch Buffer Upper Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	0216Ch-0216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_RCSUNIT

Address:	1216Ch-1216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VCSUNIT0

Address:	1A16Ch-1A16Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VECSUNIT

Address:	1C16Ch-1C16Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VCSUNIT1

Address:	2216Ch-2216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_BCSUNIT

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted batch buffer. This register follows the same rules as the BB_PREEMPT_ADDR register.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted batch buffer.

Batch Buffer Upper Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	02168h-0216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_RCSUNIT
Address:	12168h-1216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VCSUNIT0
Address:	1A168h-1A16Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VECSUNIT
Address:	1C168h-1C16Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VCSUNIT1
Address:	22168h-2216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_BCSUNIT

Description

This register specifies the upper 32 bits of the 4GB aligned base address, within the 64-bit host virtual address space of the commands being fetched from the first level batch buffer. This register has valid values only when the "Valid" bit in BB_ADDR is set to "1'.
GraphicsAddress is 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost reserved bits are ignored and MBZ.

Programming Notes

This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32]

Batch Offset Register

BB_OFFSET - Batch Offset Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000001
Access:	R/W
Size (in bits):	32

Address:	02158h-0215Bh
Name:	Batch Offset Register
ShortName:	BB_OFFSET_RCSUNIT

Address:	12158h-1215Bh
Name:	Batch Offset Register
ShortName:	BB_OFFSET_VCSUNIT0

Address:	1A158h-1A15Bh
Name:	Batch Offset Register
ShortName:	BB_OFFSET_VECSUNIT

Address:	1C158h-1C15Bh
Name:	Batch Offset Register
ShortName:	BB_OFFSET_VCSUNIT1

Address:	22158h-2215Bh
Name:	Batch Offset Register
ShortName:	BB_OFFSET_BCSUNIT

Description	Source
This register contains the offset value to be added to the Batch Buffer Start Address in the MI_BATCH_BUFFER_START command when the Enable Offset bit in MI_BATCH_BUFFER_START command is set.	

Preemptable Commands	Source
• MI_ARB_CHECK • 3D_PRIMITIVE • GPGPU_WALKER • MEDIA_STATE_FLUSH • PIPE_CONTROL (Only in GPGPU mode of pipeline selection) • MI_ATOMIC (Post Sync Operation set in GPGPU mode of pipeline selection) • MI_SEMAPHORE_SIGNAL (Post Sync Operation set in GPGPU	RenderCS

RenderCS

BB_OFFSET - Batch Offset Register

mode of pipeline selection)		
Preemptable Commands	**Source**	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
MI_ARB_CHECK	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS	

Programming Notes

On preemption occurring whith in a primary/chain batch buffer this register is loaded with the offset value of the preempted command header from the batch start address when the Enable Load is set. Preemption of 3D or GP_GPU workloads can only occur on preemptable commands. Batch buffer offset always points to the preemptable command if preempted on preemption or the immediate command following it if not preempted on preemption.

EX: Preemption occurs on 3D_PRIMITVE command

- If the 3D_PRIMTIVE command is completely processed by render pipe then the BB_OFFSET points to the command following 3D_PRIMITIVE
- If the 3D_PRIMTIVE command is not completely processed by render pipe then the BB_OFFSET points to the 3D_PRIMITIVE command.

DWord	Bit	Description
0	31:2	**Batch Buffer Offset** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned offset between the starting address of the batch buffer and where the last initiated Batch Buffer is currently fetching commands.
	1	**Reserved** Format: MBZ
	0	**Enable Load** Default Value: 1 Format: Enable **Description** If this bit is set then the Batch Buffer Offset is loaded with the preempted command offset or the following command whenever a batch buffer is ended due to a Preemptable command.

BCS Context Sizes

BCS_CXT_SIZE - BCS Context Sizes	
Register Space:	MMIO: 0/2/0
Source:	BlitterCS
Default Value:	0x00000000
Access:	Read/32 bit Write Only
Size (in bits):	32
Address:	221A8h

DWord	Bit	Description
0	31:13	**Reserved** Format: MBZ
	12:8	**BCS Context Size** Format: U5
	7:5	**Reserved** Format: MBZ
	4:0	**Execlist Context Size** Format: U5

BCS Ring Buffer Next Context ID Register

BCS_RNCID - BCS Ring Buffer Next Context ID Register

Register Space:	MMIO: 0/2/0
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	22198h-2219Fh

This register contains the *next* ring context ID associated with the ring buffer.

Programming Notes

The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that the only time a context switch can occur is when MI_ARB_CHECK enables preemption or the current context runs dry (head pointer becomes equal to tail pointer).

DWord	Bit	Description
0	63:0	**Unnamed** See Context Descriptor for BCS

BCS SW Control

BCS_SWCTRL - BCS SW Control

Register Space:	MMIO: 0/2/0
Source:	BlitterCS
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1
Address:	22200h

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask
	15:4	**Reserved** Format: MBZ
	3	**Shrink Blitter Cache** Format: U1 This bit is primarily used for validation purposes to speed up the test time. The full cache depth of 128 CLs should be used for production. This bit is part of the context save/restore. This bit only applies to the XY_FAST_COPY_BLT command. **Value** / **Name** / **Description** 0 / **[Default]** / Blitter/BCS flush will flush and invalidate all cachelines in the Blitter/BLB cache (default). 1 / / Blitter Cache depth will be shortened from 128 CLs to 16 CLs.
	2	**Not Invalidate Blitter Cache on BCS Flush** Format: U1 Programming this bit allows optimal/maximal cache hit usage, when the destination surface of a Fast Copy Blit, is to be used as the Source for a follow on Fast Copy blit, even if the destination surface is flushed out for Display coherency reasons (where the destination surface is also needed to be Displayed). Such a flush with clean cacheline state is suggested when the intermediate blit operation results are being required to maintain memory coherency. The legacy method of cache invalidation on flush can be still pursued at the end of all blit operations or when switching happens due to other prescribed legacy reasons, or when switching from the new Fast Copy Engine blit, to legacy Engine blits. This bit should be programmed set only when used with Fast Copy Blit commands. This bit is part of the context save/restore. This bit only applies to the XY_FAST_COPY_BLT command. **Value** / **Name** / **Description** 0 / **[Default]** / Blitter/BLB Cache will be 128 cache lines in depth (default).

BCS_SWCTRL - BCS SW Control

		1 | | BCS flush will put all dirty CL in the Blitter cache in the clean state. Any CL already in the clean state will remain clean.
	1	**Tile Y Destination** Format: U1 Programming this bit makes the HW treat all destination surfaces as Tile Y. This bit over-rides the setting of the destination format in the packet provided to the blitter command streamer. SW is required to flush the HW before changing the polarity of this bit. This bit is part of the context save/restore. This bit does not impact the operations of the XY_FAST_COPY_BLT command
	0	**Tile Y Source** Format: U1 Programming this bit makes the HW treat all source surfaces as Tile Y. This bit over-rides the setting of the source format in the packet provided to the blitter command streamer. SW is required to flush the HW before changing the polarity of this bit. This bit is part of the context save/restore. This bit does not impact the operations of the XY_FAST_COPY_BLT command

BITPLANE CYCLE CONTROL REGISTER

BITPLANE_CTRL - BITPLANE CYCLE CONTROL REGISTER

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32

Bitplane Cycle Control Register

DWord	Bit	Description
0	30:19	**Reserved** Default Value: 000000000000b Access: RO

Bitstream Output Bit Count for the last Syntax Element Report Register

MFC_BITSTREAM_SE_BITCOUNT_SLICE - Bitstream Output Bit Count for the last Syntax Element Report Register	
Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128D4h
Name:	SE Output Bit Count

This register stores the count of number of bits in the bitstream for the last syntax element before padding. The bit count is before the byte-aligned alignment padding insertion, but includes the stop-one-bit. This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC Bitstream Syntax Element Bit Count** Total number of bits in the bitstream output before padding. This count is updated each time the internal counter is incremented.

Bitstream Output Byte Count Per Slice Report Register

MFC_BITSTREAM_BYTECOUNT_SLICE - Bitstream Output Byte Count Per Slice Report Register

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128D0h

This register stores the count of bytes of the bitstream output. This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC Bitstream Byte Count** Total number of bytes in the bitstream output from the encoder. This count is updated for every time the internal bitstream counter is incremented.

Bitstream Output Minimal Size Padding Count Report Register

MFC_AVC_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12814h
Name:	Minimal Size Padding

This register stores the count in bytes of **minimal size padding insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC AVC MinSize Padding Count** Total number of bytes in the bitstream output contributing to minimal size padding operation. This count is updated each time when the padding count is incremented.

BLC_PWM_CTL

BLC_PWM_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	48250h-48253h
Name:	Backlight PWM Control
ShortName:	BLC_PWM_CTL
Power:	PG0
Reset:	soft

This register controls the backlight PWM logic going to the display utility pin on the CPU.

DWord	Bit	Description
0	31	**PWM Enable** This bit enables the PWM logic. Value / Name / Description: 0b / Disable / PWM disabled 1b / Enable / PWM enabled Restriction: Restriction : The display utility pin must be configured correctly to output the PWM. Program the frequency and duty cycle before enabling PWM.
	30:29	**Pipe Select** This field selects which vertical blank will be used for backlight blinking. Value / Name / Description: 00b / Pipe A / Use Pipe A 01b / Pipe B / Use Pipe B 10b / Pipe C / Use Pipe C
	28	**Blinking Enable** This bit enables backlight blinking. When enabled, the backlight will be driven on at the programmed brightness during vertical blank and driven off during vertical active. Value / Name: 0b / Disable 1b / Enable

BLC_PWM_CTL

	27	**PWM Granularity** This field controls the granularity (minimum increment) of the PWM backlight control counter.

Value	Name	Description
0b	128	PWM frequency adjustment on 128 clock increments
1b	8	PWM frequency adjustment on 8 clock increments

	26:0	**Reserved**

BLC_PWM_DATA

BLC_PWM_DATA		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	48254h-48257h	
Name:	Backlight PWM Data	
ShortName:	BLC_PWM_DATA	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:16	**Backlight Frequency** This field determines the number of time base events in total for a complete cycle of modulated backlight control. This field is programmed based on the frequency of the clock that is being used and the desired PWM frequency. This value represents the period of the PWM stream in CD clocks multiplied by 128 (default increment) or 8 (alternate increment selected by BLC_PWM_CTL PWM_Granularity).
	15:0	**Backlight Duty Cycle** This field determines the number of time base events for the active portion of the PWM backlight control. A value of zero will turn the backlight off. A value equal to the backlight modulation frequency field will be full on. Updates will take affect at the end of the current PWM cycle. This value represents the active time of the PWM stream in CD clock periods multiplied by 128 (default increment) or 8 (alternate increment selected by BLC_PWM_CTL PWM_Granularity). **Restriction** Restriction : This should never be larger than the frequency field.

Blitter MOCS Register0

BLT_MOCS_0 - Blitter MOCS Register0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0CC00h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_0 - Blitter MOCS Register0

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register1

BLT_MOCS_1 - Blitter MOCS Register1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034
Size (in bits):	32
Address:	0CC04h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_1 - Blitter MOCS Register1

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register2

BLT_MOCS_2 - Blitter MOCS Register2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038
Size (in bits):	32
Address:	0CC08h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_2 - Blitter MOCS Register2

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register3

BLT_MOCS_3 - Blitter MOCS Register3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031
Size (in bits):	32
Address:	0CC0Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 000000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_3 - Blitter MOCS Register3

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register4

BLT_MOCS_4 - Blitter MOCS Register4

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CC10h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_4 - Blitter MOCS Register4

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register5

BLT_MOCS_5 - Blitter MOCS Register5

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CC14h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_5 - Blitter MOCS Register5

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register6

BLT_MOCS_6 - Blitter MOCS Register6

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CC18h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_6 - Blitter MOCS Register6

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register7

BLT_MOCS_7 - Blitter MOCS Register7

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CC1Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_7 - Blitter MOCS Register7

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register8

BLT_MOCS_8 - Blitter MOCS Register8

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CC20h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_8 - Blitter MOCS Register8

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register9

BLT_MOCS_9 - Blitter MOCS Register9

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CC24h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_9 - Blitter MOCS Register9

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register10

BLT_MOCS_10 - Blitter MOCS Register10

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CC28h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_10 - Blitter MOCS Register10

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register11

BLT_MOCS_11 - Blitter MOCS Register11

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CC2Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_11 - Blitter MOCS Register11

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register12

BLT_MOCS_12 - Blitter MOCS Register12

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CC30h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_12 - Blitter MOCS Register12

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register13

BLT_MOCS_13 - Blitter MOCS Register13

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CC34h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_13 - Blitter MOCS Register13

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register14

BLT_MOCS_14 - Blitter MOCS Register14

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CC38h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_14 - Blitter MOCS Register14

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register15

BLT_MOCS_15 - Blitter MOCS Register15

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CC3Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_15 - Blitter MOCS Register15

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register16

BLT_MOCS_16 - Blitter MOCS Register16

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0CC40h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_16 - Blitter MOCS Register16

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register17

BLT_MOCS_17 - Blitter MOCS Register17

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0CC44h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_17 - Blitter MOCS Register17

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register18

BLT_MOCS_18 - Blitter MOCS Register18

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0CC48h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_18 - Blitter MOCS Register18

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register19

BLT_MOCS_19 - Blitter MOCS Register19

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0CC4Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_19 - Blitter MOCS Register19

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register20

BLT_MOCS_20 - Blitter MOCS Register20

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CC50h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_20 - Blitter MOCS Register20

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register21

BLT_MOCS_21 - Blitter MOCS Register21

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CC54h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_21 - Blitter MOCS Register21

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register22

BLT_MOCS_22 - Blitter MOCS Register22

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CC58h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_22 - Blitter MOCS Register22

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register23

BLT_MOCS_23 - Blitter MOCS Register23

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CC5Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_23 - Blitter MOCS Register23

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register24

BLT_MOCS_24 - Blitter MOCS Register24

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CC60h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_24 - Blitter MOCS Register24

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register25

BLT_MOCS_25 - Blitter MOCS Register25

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CC64h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_25 - Blitter MOCS Register25

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register26

BLT_MOCS_26 - Blitter MOCS Register26

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CC68h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_26 - Blitter MOCS Register26

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register27

BLT_MOCS_27 - Blitter MOCS Register27

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CC6Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_27 - Blitter MOCS Register27

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register28

BLT_MOCS_28 - Blitter MOCS Register28

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CC70h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_28 - Blitter MOCS Register28

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register29

BLT_MOCS_29 - Blitter MOCS Register29

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CC74h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_29 - Blitter MOCS Register29

6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register30

BLT_MOCS_30 - Blitter MOCS Register30

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CC78h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_30 - Blitter MOCS Register30

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register31

BLT_MOCS_31 - Blitter MOCS Register31

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CC7Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_31 - Blitter MOCS Register31

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register32

BLT_MOCS_32 - Blitter MOCS Register32

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0CC80h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_32 - Blitter MOCS Register32

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register33

BLT_MOCS_33 - Blitter MOCS Register33

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0CC84h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_33 - Blitter MOCS Register33

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register34

BLT_MOCS_34 - Blitter MOCS Register34

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0CC88h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_34 - Blitter MOCS Register34

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register35

BLT_MOCS_35 - Blitter MOCS Register35

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0CC8Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_35 - Blitter MOCS Register35

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register36

BLT_MOCS_36 - Blitter MOCS Register36

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CC90h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_36 - Blitter MOCS Register36

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register37

BLT_MOCS_37 - Blitter MOCS Register37

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CC94h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_37 - Blitter MOCS Register37

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register38

BLT_MOCS_38 - Blitter MOCS Register38

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CC98h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_38 - Blitter MOCS Register38

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register39

BLT_MOCS_39 - Blitter MOCS Register39

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CC9Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_39 - Blitter MOCS Register39

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register40

BLT_MOCS_40 - Blitter MOCS Register40

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CCA0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_40 - Blitter MOCS Register40

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register41

BLT_MOCS_41 - Blitter MOCS Register41

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CCA4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_41 - Blitter MOCS Register41

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register42

BLT_MOCS_42 - Blitter MOCS Register42

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CCA8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_42 - Blitter MOCS Register42

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register43

BLT_MOCS_43 - Blitter MOCS Register43

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CCACh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_43 - Blitter MOCS Register43

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register44

BLT_MOCS_44 - Blitter MOCS Register44

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CCB0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_44 - Blitter MOCS Register44

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register45

BLT_MOCS_45 - Blitter MOCS Register45

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CCB4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_45 - Blitter MOCS Register45

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register46

BLT_MOCS_46 - Blitter MOCS Register46

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CCB8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_46 - Blitter MOCS Register46

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register47

BLT_MOCS_47 - Blitter MOCS Register47

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CCBCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 000000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_47 - Blitter MOCS Register47

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register48

BLT_MOCS_48 - Blitter MOCS Register48

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0CCC0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_48 - Blitter MOCS Register48

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register49

BLT_MOCS_49 - Blitter MOCS Register49

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0CCC4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_49 - Blitter MOCS Register49

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register50

BLT_MOCS_50 - Blitter MOCS Register50

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0CCC8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_50 - Blitter MOCS Register50

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register51

BLT_MOCS_51 - Blitter MOCS Register51

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0CCCCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_51 - Blitter MOCS Register51

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register52

BLT_MOCS_52 - Blitter MOCS Register52

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0CCD0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_52 - Blitter MOCS Register52

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register53

BLT_MOCS_53 - Blitter MOCS Register53

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CCD4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_53 - Blitter MOCS Register53

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register54

BLT_MOCS_54 - Blitter MOCS Register54

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CCD8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_54 - Blitter MOCS Register54

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register55

BLT_MOCS_55 - Blitter MOCS Register55

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0CCDCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_55 - Blitter MOCS Register55

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register56

BLT_MOCS_56 - Blitter MOCS Register56

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CCE0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_56 - Blitter MOCS Register56

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register57

BLT_MOCS_57 - Blitter MOCS Register57

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CCE4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_57 - Blitter MOCS Register57

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register58

BLT_MOCS_58 - Blitter MOCS Register58

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]]
Size (in bits):	32
Address:	0CCE8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_58 - Blitter MOCS Register58

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register59

BLT_MOCS_59 - Blitter MOCS Register59

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0CCECh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_59 - Blitter MOCS Register59

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register60

BLT_MOCS_60 - Blitter MOCS Register60

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0CCF0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_60 - Blitter MOCS Register60

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register61

BLT_MOCS_61 - Blitter MOCS Register61

Register Space:	MMIO: 0/2/0
Source:	BSpec
	0x00000033 [KBL]
Size (in bits):	32
Address:	0CCF4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_61 - Blitter MOCS Register61

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register62

BLT_MOCS_62 - Blitter MOCS Register62

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0CCF8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_62 - Blitter MOCS Register62

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter MOCS Register63

BLT_MOCS_63 - Blitter MOCS Register63

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0CCFCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

BLT_MOCS_63 - Blitter MOCS Register63

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Blitter TLB Control Register

BTCR - Blitter TLB Control Register

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0426Ch

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

BLT Context Element Descriptor (High Part)

BLT_CTX_EDR_H - BLT Context Element Descriptor (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04504h	
DWord	**Bit**	**Description**
0	31:0	**BLT Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W

BLT Context Element Descriptor (Low Part)

BLT_CTX_EDR_L - BLT Context Element Descriptor (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000009	
Size (in bits):	32	
Address:	04500h	
DWord	**Bit**	**Description**
0	31:0	**BLT Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W

BLT Fault Counter

BLT_FAULT_CNTR - BLT Fault Counter		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	045B8h	
DWord	**Bit**	**Description**
0	31:0	**BLT Fault Counter** Default Value: 00000000h Access: RO This counter only applies to advance context when fault and stream mode is selected.

BLT Fixed Counter

BLT_FIXED_CNTR - BLT Fixed Counter		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	045BCh	
DWord	**Bit**	**Description**
0	31:0	**BLT Fixed Counter** Default Value: 00000000h Access: RO This counter only applies to advance context when fault and stream mode is selected.

BLT PDP0/PML4/PASID Descriptor (High Part)

BLT_CTX_PDP0_H - BLT PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0450Ch

DWord	Bit	Description
0	31:0	**BLT PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W

BLT PDP0/PML4/PASID Descriptor (Low Part)

BLT_CTX_PDP0_L - BLT PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04508h

DWord	Bit	Description
0	31:0	**BLT PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W

BLT PDP1 Descriptor Register (High Part)

BLT_CTX_PDP1_H - BLT PDP1 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04514h	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP1 Descriptor Register (Low Part)

BLT_CTX_PDP1_L - BLT PDP1 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04510h	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

BLT PDP2 Descriptor Register (High Part)

BLT_CTX_PDP2_H - BLT PDP2 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0451Ch	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP2 Descriptor Register (Low Part)

BLT_CTX_PDP2_L - BLT PDP2 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04518h	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

BLT PDP3 Descriptor Register (High Part)

BLT_CTX_PDP3_H - BLT PDP3 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04524h	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP3 Descriptor Register (Low Part)

BLT_CTX_PDP3_L - BLT PDP3 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04520h	
DWord	**Bit**	**Description**
0	31:0	**BLT PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

Boolean_Counter_B0

OAPERF_B0 - Boolean_Counter_B0	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02920h

This register enables the current live value of performance counter B0 to be read. Since what conditions cause B0 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B1

OAPERF_B1 - Boolean_Counter_B1	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02924h

This register enables the current live value of performance counter B1 to be read. Since what conditions cause B1 to increment are defined by the programming of CEC1-0/CEC1-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B2

OAPERF_B2 - Boolean_Counter_B2	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02928h

This register enables the current live value of performance counter B2 to be read. Since what conditions cause B2 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B3

OAPERF_B3 - Boolean_Counter_B3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0292Ch

This register enables the current live value of performance counter B3 to be read. Since what conditions cause B3 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B4

OAPERF_B4 - Boolean_Counter_B4	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02930h

This register enables the current live value of performance counter B4 to be read. Since what conditions cause B4 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B5

OAPERF_B5 - Boolean_Counter_B5

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02934h

This register enables the current live value of performance counter B5 to be read. Since what conditions cause B5 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B6

OAPERF_B6 - Boolean_Counter_B6	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02938h

This register enables the current live value of performance counter B6 to be read. Since what conditions cause B6 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B7

OAPERF_B7 - Boolean_Counter_B7

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0293Ch

This register enables the current live value of performance counter B7 to be read. Since what conditions cause B7 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

BOOT VECTOR

BOOTMSG - BOOT VECTOR

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	08504h

Boot Message Register
This register gets locked by the Hardware once written and is cleared only during the reset. This is extra protection given against Illegal Programming.

DWord	Bit	Description
0	31:0	**Boot Vector Message** Access: R/W Lock Boot vector is pass through. MBC gets the boot message from GPMunit and forwards it to MSQC. Breakdown of message is done in MSQC. Details: if b[26] = 1 C6SliceA = b[20:17]; C6SliceB= d[13:10] C6Way = 0 C6Area = 0 if b[26] = 0 C6Way = b[25:21], C6Slice = d[20:17]; C6Area = d[17:10] Context Restore = b[6] Reset Type = b[6:5] Ring Stop ID = b[4:0]

BTB Not Consumed By RCS

BTP_PRODUCE_COUNT - BTB Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02480h

This register keeps track of the outstanding BTP produced by RS which are not yet consumed by Render Command Streamer.
This register is part of the render context save and restore.

Programming Notes

This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**BTP Produce Count** This register keeps track of the outstanding BTP produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

BTP Commands Parsed By RCS

BTP_PARSE_COUNT - BTP Commands Parsed By RCS

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02490h

This register keeps track of the BTP commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has BTP Pool Alloc Valid. BTP parse count should be less then equal to the BTP produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. BTP Parse count is subtracted from the BTP Produce Count upon parsing 3D_PRIMITIVE command. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**BTP Parse Count** This register keeps track of the BTP commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has BTP Pool Alloc Valid. BTP parse count should be less then equal to the BTP produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. BTP Parse count is subtracted from the BTP Produce Count upon parsing 3D_PRIMITIVE command.

Built In Self Test

BIST_0_2_0_PCI - Built In Self Test		
Register Space:	PCI: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	8	
Address:	0000Fh	
This register is used for control and status of Built In Self Test (BIST).		
DWord	**Bit**	**Description**
0	7	**BIST Supported** Default Value: 0b Access: RO BIST is not supported. This bit is hardwired to 0.
	6:0	**Reserved** Format: MBZ

Cache Line Size

CLS_0_2_0_PCI - Cache Line Size		
Register Space:	PCI: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	8	
Address:	0000Ch	
DWord	**Bit**	**Description**
0	7:0	**Cache Line Size** Default Value: 00000000b Access: R/W This register is not reset by FLR. Implemented for PCIe compliant devices for legacy compatibility but has no effect on any PCIe device behavior.

Cache Mode Register 0

CACHE_MODE_0 - Cache Mode Register 0

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000 [KBL]
Access:	R/W
Size (in bits):	32
Address:	07000h

Description

This register is used to control the operation of the Render and Sampler L2 Caches. All reserved bits are implemented as read/write.
Before changing the value of this register, GFX pipeline must be idle i.e. full flush is required.
This Register is saved and restored as part of Context.

RegisterType = MMIO_SVL

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask[15:0] A 1 in a bit in this field allows the modification of the corresponding bit in Bits 15:0.
	15	**Sampler L2 Disable** Access: r/w Format: Disable 0h **[Default]** – Sampler L2 Cache Enabled. 1h – Sampler L2 Cache Disabled. All accesses are treated as misses.
	14:12	**MSAA Compression Plane Number Threshold for eLLC** Access: r/w 0h threshold0 **[Default]** – Cache only planeID = 0 in eLLC. 1h threshold1 – Cache only planeID = 0, 1 in eLLC. 2h threshold2 – Cache only planeID = 0..2 in eLLC. 3h threshold3 – Cache only planeID = 0..3 in eLLC. 4h threshold4 – Cache only planeID = 0..4 in eLLC. 5h threshold5 – Cache only planeID = 0..5 in eLLC.

CACHE_MODE_0 - Cache Mode Register 0

6h	threshold6	Cache only planeID = 0..6 in eLLC.
7h	threshold7	Cache only planeID = 0..7 in eLLC.

Programming Notes
This bit-field is programmed based on MSAA. When MSAA compression is enabled, these settings affect HW, else it is ignored. For 16X MSAA only lower 8 planes can be cached in eLLC.

11 **Sampler Set Remmapping for 3D Disable**

Access:	r/w

Value	Name	Description
0h	Enable Set Remap **[Default]**	Set remapping for 3d enabled
1h	Disable Set Remap	Set remapping for 3d disabled

10 **Reserved**

Access:	r/w
Format:	PBC

9 **Sampler L2 TLB Prefetch Enable**

Access:	r/w

Value	Name	Description
0h	**[Default]**	TLB Prefetch Disabled
1h		TLB Prefetch Enabled

8 **Reserved**

7:6 **Sampler L2 Request Arbitration**

Access:	r/w
Format:	U2

Value	Name	Description
00b		Round Robin
01b		Fetch are Highest Priority
10b		Constants are Highest Priority
11b		Reserved

5 **STC PMA Optimization Enable**

Access:	r/w
Format:	Enable

Clearing this bit will force the STC cache to wait for pending retirement of pixels at the HZ-read stage and do the STC-test for Non-promoted, R-computed and Computed depth modes instead of postponing the STC-test to RCPFE.

CACHE_MODE_0 - Cache Mode Register 0

Value	Name	Description
0h	Disable **[Default]**	STC PMA optimization is disabled.
1h	Enable	STC PMA optimization is enabled.

4 **RCC Eviction Policy**

Access:	r/w
Format:	Disable

If this bit is set, RCCunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.

Programming Notes
If this bit is set to "1", bit 7 of 0x7010h must also be set to "1".

3 **Reserved**

2 **Hierarchical Z RAW Stall Optimization Disable**

Access:	r/w
Format:	U1

The Hierarchical Z RAW Stall Optimization allows non-overlapping polygons in the same 8x4 pixel/sample area to be processed without stalling waiting for the earlier ones to write to Hierarchical Z buffer.

Value	Name	Description
0h	Enable **[Default]**	Enables the hierarchical Z RAW Stall Optimization.
1h	Disable	Disables the hierarchical Z RAW Stall Optimization.

Programming Notes
This bit must be set to 1 to disable the Hierarchical Z RAW stall optimization.

1 **Disable clock gating in the pixel backend**

Access:	r/w
Format:	Disable

MCL related clock gating is disabled in the pixel backend. Before setting this bit to 1, the instruction/state caches must be invalidated. [DevGT:{WKA}]

0 **Null tile fix disable**

Access:	r/w

Instead of dropping non dirty cachelines at alloc point, we allow the cacheline till the mem wrbk point, so that null status can be reset.

Value	Name	Description
0	**[Default]**	Null tile fix enabled
1		Null tile fix disable

Cache Mode Register 1

CACHE_MODE_1 - Cache Mode Register 1

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00002980 [KBL]
Access:	R/W
Size (in bits):	32
Address:	07004h

Description
RegisterType: MMIO_SVL
Before changing the value of this register, GFX pipeline must be idle; i.e., full flush is required. This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Mask: MASK Format: Mask[15:0] Must be set to modify corresponding data bit. Reads to this field returns zero.
	15	**Color Compression Disable** Access: r/w Setting this bit causes Lossless Render Target Color Compression to be disabled in Classic Clear (1x) Mode of Operation. Default value, i.e. resetting this bit, Enables Color Compression in Classic Clear Mode (1x) when CCS is Enabled. Value / Name: 0h: Enable **[Default]** 1h: Disable **Programming Notes** The Below programming forces Color Compression to be disabled for MSAA modes explicitly as a HW WA. When switching from 1x ==> MSAA. Program this bit to 1 When switching from MSAA ==> 1x. Program this bit to 0
	14	**Blend Optimization Fix Disable** This bit when reset, enables blend optimization fix. If this bit is set, it disables the blend optimization fix and may exhibit corruption on alpha components in the render target under some conditions.

CACHE_MODE_1 - Cache Mode Register 1

13 **NP EARLY Z FAILS DISABLE**

Access:	r/w

Value	Name	Description
0h	Disable	When NP PMA FIX ENABLE = 1, clearing this bit disables IZ to conservatively fail pixels.
1h	Enable **[Default]**	When NP PMA FIX ENABLE = 1, IZ does conservatively fail any NP pixels.

12 **HIZ Eviction Policy**

Access:	r/w
Format:	U1

If this bit is set, Hizunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates the non-LRA eviction policy. For performance reasons, this bit must be reset.

Value	Name	Description
0h	**[Default]**	Non-LRA eviction Policy
1h		LRA eviction Policy

Programming Notes
If this bit is set to "1", bit 3 of 0x7010h must also be set to "1"

11 **NP PMA FIX ENABLE**

Access:	r/w

Value	Name	Description
0h	Disable	Enables stalling PMA behavior for NP depth pixels in the early depth pipeline. (Legacy behavior)
1h	Enable **[Default]**	Enables non-stalling PMA behavior for NP depth pixels in the early depth pipeline.

Programming Notes
PMA Optimization Enable bit can be programmed to 0 to disable this optimization.

10 **Reserved**

9 **MSC RAW Hazard Avoidance Bit**

Access:	r/w
Format:	Enable

When this field is set, MSC will enable RAW Hazard prevention mechanism, when lossless compression is enabled.

Value	Name	Programming Notes
0h	**[Default]**	

CACHE_MODE_1 - Cache Mode Register 1

1h		This field should be programmed to 1 only if need arise to avoid RAW hazard when lossless compression is enabled

8:7 **Sampler Cache Set XOR selection**

Access:	r/w
Format:	U2

These bits have an impact only when the Sampler cache is configured in 16 way set associative mode. If the cache is being used for immediate data or for blitter data these bits have no effect.

Value	Name	Description
00b	None	No XOR.
01b	Scheme 1	New_set_mask[3:0] = Tiled_address[16:13]. New_set[3:0] less than or = New_set_mask[3:0] ^Old_set[3:0]. Rationale: These bits can distinguish among 16 different equivalent classes of virtual pages. These bits also represent the lsb for tile rows ranging from a pitch of 1 tile to 16 tiles.
10b	Scheme 2	New_set_mask[3] = Tiled_address[17] ^ Tiled_address[16]. New_set_mask[2] = Tiled_address[16] ^ Tiled_address[15]. New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14]. New_set_mask[0] = Tiled_address[14] ^ Tiled_address[13]. New_set[3:0] less than or = New_set_mask[3:0] ^ Old_set[3:0]. Rationale: More bits on each XOR can give better statistical uniformity on sets and since two lsbs are taken for each tile row size, it reduces the chance of aliasing on sets.
11b	Scheme 3 **[Default]**	New_set_mask[3] = Tiled_address[22] ^ Tiled_address[21] ^ Tiled_address[20] ^ Tiled_address[19]. New_set_mask[2] = Tiled_address[18] ^ Tiled_address[17] ^ Tiled_address[16]. New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14]. New_set_mask[0] = Tiled_address[13]. New_set[3:0] less than or = New_set_mask[3:0] ^ Old_set[3:0]. Rationale: More bits on each XOR can give better statistical uniformity on sets and since each XOR has different bits, it reduces the chance of aliasing on sets even more.

6 **4X4 RCPFE-STC Optimization Disable**

Access:	r/w
Format:	Disable

Value	Name	Description
0h	**[Default]**	Enables two contiguous 4x2s to be collected as 4X4 access for STC interface. This allows for less bank collision and less RAM power on STC.
1h		Disables this optimization and therefore only one valid 4x2 is sent to STC on the 4X4 interface.

CACHE_MODE_1 - Cache Mode Register 1

Restriction
Restriction This bit must be set.

5 MCS Cache Disable

Access:	r/w
Format:	Disable

For Programming restrictions please refer to the 3D Pipeline.

Value	Name	Description
0h	**[Default]**	MCS cache enabled. It allows RTs with MCS buffer enabled to be rendered using either MSAA compression for MSRT OR with color clear feature for non MSRT.
1h		MCS cache is disabled. Hence no MSAA compression for MSRT and no color clear for non-MSRT.

4 Float Blend Optimization Enable

Access:	r/w
Format:	Enable

Value	Name	Description
0h	**[Default]**	Disables blend optimization for floating point RTs.
1h		Enables blend optimization for floating point RTs.

3 Depth Read Hit Write-Only Optimization Disable

Access:	r/w
Format:	Disable

Value	Name	Description
0h	**[Default]**	Read Hit Write-only optimization is enabled in the Depth cache (RCZ).
1h		Read Hit Write-only optimization is disabled in the Depth cache (RCZ).

2 RCZ Read after expansion control fix 2

Access:	r/w
Format:	Enable

Value	Name	Description
0h	**[Default]**	RCZ will suppress the read request to memory if it was allocated as a expansion Cacheline
1h		RCZ will always issue a read request to memory, even if it was previously allocated as expansion Cacheline

CACHE_MODE_1 - Cache Mode Register 1

	1	**Partial Resolve Disable in VC** Access: r/w Format: Disable **Value / Name / Description** 0h **[Default]** Partial resolve in the Victim Cache enabled. 1h Partial evictions of cachelines with compression disabled from pixel backend (RCC) will not be resolved in the Victim Cache in CC in order to extend the resolve queue in GAM. Compression enabled cases will ignore this bit and will always be resolved in the VC.
	0	**Reserved**

Capabilities A

CAPID0_A_0_0_0_PCI - Capabilities A

Register Space:	PCI: 0/0/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	000E4h

DWord	Bit	Description
0	31	**Display HD Audio Disable** Default Value: 0b Access: R/W Key Firmware Only
	30	**PEG12 Disable** Default Value: 0b Access: R/W Key Firmware Only
	29	**PEG11 Disable** Default Value: 0b Access: R/W Key Firmware Only
	28	**PEG10 Disable** Default Value: 0b Access: R/W Key Firmware Only
	27	**PCI Express Link Width Upconfig Disable** Default Value: 0b Access: R/W Firmware Only
	26	**DMI Width** Default Value: 0b Access: R/W Firmware Only
	25	**ECC Disable** Default Value: 0b Access: R/W Firmware Only
	24	**Force DRAM ECC Enabled** Default Value: 0b Access: R/W Firmware Only

CAPID0_A_0_0_0_PCI - Capabilities A

23	**VTd Disable**	
	Default Value:	0b
	Access:	R/W Key Firmware Only
	0: Enable VTd 1: Disable VTd	
22	**DMI Gen 2 Disable**	
	Default Value:	0b
	Access:	R/W Firmware Only
21	**PEG Gen 2 Disable**	
	Default Value:	0b
	Access:	R/W Firmware Only
20:19	**DDR Size**	
	Default Value:	00b
	Access:	R/W Firmware Only
18	**SPARE18**	
	Default Value:	0b
	Access:	R/W Firmware Only
17	**Disable 1N Mode**	
	Default Value:	0b
	Access:	R/W Firmware Only
16	**Full ULT Fuse Read Disable**	
	Default Value:	0b
	Access:	R/W Firmware Only
15	**Camarillo Device Disable**	
	Default Value:	0b
	Access:	R/W Key Firmware Only
14	**2 DIMMS per Channel Disable**	
	Default Value:	0b
	Access:	R/W Firmware Only
13	**X2APIC Enabled**	
	Default Value:	0b
	Access:	R/W Firmware Only
12	**Performance Dual Channel Disable**	
	Default Value:	0b
	Access:	R/W Firmware Only

CAPID0_A_0_0_0_PCI - Capabilities A

	11	**Internal Graphics Disable** Default Value: 0b Access: R/W Key Firmware Only 0b: There is a graphics engine within this CPU. Internal Graphics Device (Device 2) is enabled and all of its memory and I/O spaces are accessable. Configuration cycles to Device 2 will be completed within the CPU. All non-SMM memory and IO accesses to VGA will be handled based on Memory and IO enables of Device 2 and IO registers within Device 2 and VGA Enable of the PCI to PCI bridge control (If PCI Express GFX attach is supported). A selected amount of Graphics Memory space is pre-allocated from the main memory based on Graphics Mode Select (GMS in the GGC Register). Graphics Memory is pre-allocated above TSEG Memory. 1b: There is no graphics engine within this CPU. Internal Graphics Device (Device 2) and all of its memory and I/O functions are disabled. Configuration cycle targeted to Device 2 will be passed on. All non-SMM memory and IO accesses to VGA will be handled based on VGA Enable of the PCI to PCI bridge control. Device 2 is disabled and hidden.
	10	**Reserved**
	9	**Reserved**
	8	**SPARE8** Default Value: 0b Access: R/W Firmware Only
	7:4	**Compatibility Rev ID** Default Value: 0000b Access: R/W Firmware Only This is an 8-bit value that indicates the revision identification number for the Host Device 0.
	3	**DDR Overclocking** Default Value: 0b Access: R/W Firmware Only
	2	**IA Overclocking Enabled by DSKU** Default Value: 0b Access: R/W Firmware Only
	1	**DDR Write VRef** Default Value: 0b Access: R/W Firmware Only
	0	**DDR3L Enable** Default Value: 0b Access: R/W Firmware Only

Capabilities B

CAPID0_B_0_0_0_PCI - Capabilities B

Register Space:	PCI: 0/0/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	000E8h

DWord	Bit	Description
0	31	**IMGU Disable** Default Value: 0b Access: R/W Key Firmware Only
	30	**SPARE30** Default Value: 0b Access: R/W Firmware Only
	29	**IA Overclocking Enable** Default Value: 0b Access: R/W Firmware Only
	28	**SMT Capability** Default Value: 0b Access: R/W Firmware Only
	27:25	**Cache Size Capability** Default Value: 000b Access: R/W Firmware Only
	24	**SPARE24** Default Value: 0b Access: R/W Firmware Only
	23:21	**DDR3 Maximum Frequency Capability with 100 Memory** Default Value: 000b Access: R/W Firmware Only
	20	**Gen3 Disable Fuse for PCIe PEG Controllers** Default Value: 0b Access: R/W Firmware Only
	19	**Package Type** Default Value: 0b Access: R/W Firmware Only

CAPID0_B_0_0_0_PCI - Capabilities B

	18	**Additive Graphics Enabled** Default Value: 0b Access: R/W Firmware Only 0 - Additive Graphics Disabled 1- Additive Graphics Enabled
	17	**Additive Graphics Capable** Default Value: 0b Access: R/W Firmware Only 0 - Capable of Additive Graphics 1 - Not capable of Additive Graphics
	16	**Primary PEG Port x16 Disable** Default Value: 0b Access: R/W Firmware Only
	15	**DMIG3 Disable** Default Value: 0b Access: R/W Firmware Only
	14:12	**SPARE14_12** Default Value: 000b Access: R/W Firmware Only
	11	**Reserved**
	10:9	**SPARE10_9** Default Value: 00b Access: R/W Firmware Only
	8	**GMM Disable** Default Value: 0b Access: R/W Key Firmware Only
	7	**Reserved**
	6:4	**DDR3 Maximum Frequency Capability** Default Value: 000b Access: R/W Firmware Only
	3	**SPARE3** Default Value: 0b Access: R/W Firmware Only

CAPID0_B_0_0_0_PCI - Capabilities B		
	2	**DDR4 DSKU Enable** Default Value: 0b Access: R/W Firmware Only
	1	**Dual PEG Force x1 when VGA Enabled** Default Value: 0b Access: R/W Firmware Only
	0	**Single PEG Force x1 when VGA Enabled** Default Value: 0b Access: R/W Firmware Only

Capabilities Control

CAPCTRL0_0_2_0_PCI - Capabilities Control

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x0000010C
Size (in bits):	16
Address:	00042h

DWord	Bit	Description
0	11:8	**CAPID Version** Default Value: 0001b Access: RO This field is hardwired to the value 1h to identify the first revision of the CAPID register definition.
	7:0	**CAPID Length** Default Value: 00001100b Access: RO This field is hardwired to the value 0Ch to indicate the structure length (12 bytes).

Capabilities Pointer

CAPPOINT_0_2_0_PCI - Capabilities Pointer

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000040
Size (in bits):	8
Address:	00034h

This register points to a linked list of capabilities implemented by this device.

DWord	Bit	Description
0	7:0	**Capabilities Pointer Value** Default Value: 01000000b Access: RO This field contains an offset into the function's PCI Configuration Space for the first item in the New Capabilities Linked List, the CAPID0 register at offset 40h.

Capability Identifier

CAPID0_0_2_0_PCI - Capability Identifier

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00007009
Size (in bits):	16
Address:	00040h

DWord	Bit	Description
0	15:8	**Next Capability Pointer** Default Value: 01110000b Access: RO This field is hardwired to point to the next PCI Capability structure, the PCIe Capabilities structure at 70h.
	7:0	**Capability Identifier** Default Value: 00001001b Access: RO This field is hardwired to the value 09h to identify the CAP_ID assigned by the PCI SIG for vendor dependent capability pointers.

CDCLK_CTL

CDCLK_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x080002A1 [KBL]
Access:	R/W
Size (in bits):	32
Address:	46000h-46003h
Name:	CD Clock Control
ShortName:	CDCLK_CTL
Power:	PG0
Reset:	global

This register is not reset by the device 2 FLR.

Restriction
Restriction : These fields should only be changed as part of the Display Sequences for Changing CD Clock Frequency.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:26	**CD Frequency Select** This field, together with the DPLL0 VCO setting, selects the frequency for CD clock. The DPLL0 VCO setting is programmed through DPLL_CTRL1.

Value	Name	Description
00b	450 or 432 MHz	If DPLL0 VCO 8100 (450 MHz CD, 900 MHz CD2X); If DPLL0 VCO 8640 (432 MHz CD, 864 MHz CD2X);
01b	540 MHz	540 MHz CD, 1080 MHz CD2X
10b	337.5 or 308.57 MHz **[Default]**	If DPLL0 VCO 8100 (337.5 MHz CD, 675 MHz CD2X); If DPLL0 VCO 8640 (308.57 MHz CD, 617.14 MHz CD2X);
11b	675 or 617.14 MHz	If DPLL0 VCO 8100 (675 MHz CD, 1350 MHz CD2X); If DPLL0 VCO 8640 (617.14 MHz CD, 1234.28 MHz CD2X);

Restriction
Restriction : The Display CD Clock Frequency Limit fuse indicates the maximum allowed CD clock frequency. Software must not select any frequency higher than the maximum that is allowed. If software incorrectly selects a higher frequency, hardware will override to the lowest frequency.

	25:20	**Reserved** Format: MBZ

CDCLK_CTL

	19	**Reserved**
	18	**Reserved**
	17	**Reserved**
	16	**SSA Precharge Enable** This field is unused. **Value** / **Name**: 0b – Disable
	15:11	**Reserved** Format: MBZ
	10:0	**CD Frequency Decimal** Format: U10.1 This field selects the decimal value of the frequency for CD clock, which is used to generate divided down clocks for some display engine timers. This value is represented in a 10.1 format with 10 integer bits and 1 fractional bit. Program this field to match the CD frequency chosen by the CD Frequency Select (considering the DPLL0 VCO that is being used), minus one.

Value	Name
01 0011 0011 1b	308.57 MHz CD
01 0101 0000 1b	337.5 MHz CD **[Default]**
01 1010 1111 0b	432 MHz CD
01 1100 0001 0b	450 MHz CD
10 0001 1011 0b	540 MHz CD
10 0110 1000 0b	617.14 MHz CD
10 1010 0010 0b	675 MHz CD

CGE_CTRL

CGE_CTRL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Address:	49080h-49083h
Name:	Pipe Color Gamut Enhancement Control
ShortName:	CGE_CTRL_A
Power:	PG1
Reset:	soft

Address:	49180h-49183h
Name:	Pipe Color Gamut Enhancement Control
ShortName:	CGE_CTRL_B
Power:	PG2
Reset:	soft

Address:	49280h-49283h
Name:	Pipe Color Gamut Enhancement Control
ShortName:	CGE_CTRL_C
Power:	PG2
Reset:	soft

DWord	Bit	Description
0	31	**CGE Enable** This bit enables the Color Gamut Enhancement logic. Value: 0b — Disable Value: 1b — Enable
	30:0	**Reserved** Format: MBZ

CGE_WEIGHT

CGE_WEIGHT	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	R/W
Size (in bits):	160
Address:	49090h-490A3h
Name:	Pipe Color Gamut Enhancement Weights
ShortName:	CGE_WEIGHT_A
Power:	PG1
Reset:	soft
Address:	49190h-491A3h
Name:	Pipe Color Gamut Enhancement Weights
ShortName:	CGE_WEIGHT_B
Power:	PG2
Reset:	soft
Address:	49290h-492A3h
Name:	Pipe Color Gamut Enhancement Weights
ShortName:	CGE_WEIGHT_C
Power:	PG2
Reset:	soft

These are the weights contained in the lookup up table (LUT) used in pipe color gamut enhancement.
LUT index 0 contains the weight for the least saturated colors, and LUT index 16 contains the weight for the most saturated colors.
Weight values can range from 00000b (100% of the enhanced output color is from the pipe gamma and CSC output corrected color) to 100000b (100% of the enhanced output color is from the pipe gamma and CSC input color).

Restriction
Restriction : The weight values should only be changed while color gamut enhancement is disabled, otherwise screen artifacts may show temporarily.

DWord	Bit	Description
0	31:30	**Reserved** Format: MBZ
	29:24	**CGE Weight Index 3** This is the weight value for this color gamut enhancement LUT index.

CGE_WEIGHT		
	23:22	**Reserved** Format: MBZ
	21:16	**CGE Weight Index 2** This is the weight value for this color gamut enhancement LUT index.
	15:14	**Reserved** Format: MBZ
	13:8	**CGE Weight Index 1** This is the weight value for this color gamut enhancement LUT index.
	7:6	**Reserved** Format: MBZ
	5:0	**CGE Weight Index 0** This is the weight value for this color gamut enhancement LUT index.
1	31:30	**Reserved** Format: MBZ
	29:24	**CGE Weight Index 7** This is the weight value for this color gamut enhancement LUT index.
	23:22	**Reserved** Format: MBZ
	21:16	**CGE Weight Index 6** This is the weight value for this color gamut enhancement LUT index.
	15:14	**Reserved** Format: MBZ
	13:8	**CGE Weight Index 5** This is the weight value for this color gamut enhancement LUT index.
	7:6	**Reserved** Format: MBZ
	5:0	**CGE Weight Index 4** This is the weight value for this color gamut enhancement LUT index.
2	31:30	**Reserved** Format: MBZ
	29:24	**CGE Weight Index 11** This is the weight value for this color gamut enhancement LUT index.
	23:22	**Reserved** Format: MBZ
	21:16	**CGE Weight Index 10** This is the weight value for this color gamut enhancement LUT index.

CGE_WEIGHT		
	15:14	**Reserved** Format: MBZ
	13:8	**CGE Weight Index 9** This is the weight value for this color gamut enhancement LUT index.
	7:6	**Reserved** Format: MBZ
	5:0	**CGE Weight Index 8** This is the weight value for this color gamut enhancement LUT index.
3	31:30	**Reserved** Format: MBZ
	29:24	**CGE Weight Index 15** This is the weight value for this color gamut enhancement LUT index.
	23:22	**Reserved** Format: MBZ
	21:16	**CGE Weight Index 14** This is the weight value for this color gamut enhancement LUT index.
	15:14	**Reserved** Format: MBZ
	13:8	**CGE Weight Index 13** This is the weight value for this color gamut enhancement LUT index.
	7:6	**Reserved** Format: MBZ
	5:0	**CGE Weight Index 12** This is the weight value for this color gamut enhancement LUT index.
4	31:6	**Reserved** Format: MBZ
	5:0	**CGE Weight Index 16** This is the weight value for this color gamut enhancement LUT index.

Class Code

CC_0_2_0_PCI - Class Code

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00030000
Size (in bits):	24
Address:	00009h

This register contains the device programming interface information related to the Sub-Class Code and Base Class Code definition for the IGD. This register also contains the Base Class Code and the function sub-class in relation to the Base Class Code.

DWord	Bit	Description
0	23:16	**Base Class Code** Default Value: 00000011b Access: RO Variant This is an 8-bit value that indicates the base class code. When MGGC0[VAMEN] is 0 this code has the value 03h, indicating a Display Controller. When MGGC0[VAMEN] is 1 this code has the value 04h, indicating a Multimedia Device.
	15:8	**Sub-Class Code** Default Value: 00000000b Access: RO Variant When MGGC0[VAMEN] is 0 this value will be determined based on Device 0 GGC register, GMS and IVD fields. 00h: VGA compatible 80h: Non VGA (GMS = "00h" or IVD = "1b") When MGGC0[VAMEN] is 1, this value is 80h, indicating other multimedia device.
	7:0	**Programming Interface** Default Value: 00000000b Access: RO When MGGC0[VAMEN] is 0 this value is 00h, indicating a Display Controller. When MGGC0[VAMEN] is 1 this value is 00h, indicating a NOP.

Clipper Invocation Counter

CL_INVOCATION_COUNT - Clipper Invocation Counter	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02338h

This register stores the count of objects entering the Clipper stage. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**CL Invocation Count Report UDW** Number of objects entering the clipper stage. Updated only when Statistics Enable is set in CLIP_STATE (see the Clipper Chapter in the 3D Volume.)
	31:0	**CL Invocation Count Report LDW** Number of objects entering the clipper stage. Updated only when Statistics Enable is set in CLIP_STATE (see the Clipper Chapter in the 3D Volume.)

Clipper Primitives Counter

CL_PRIMITIVES_COUNT - Clipper Primitives Counter	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02340h

This register reflects the total number of primitives that have been output by the clipper. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**Clipped Primitives Output Count UDW** Total number of primitives output by the clipper stage. This count is updated for every primitive output by the clipper stage, as long as Statistics Enable is set in SF_STATE (see the Clipper and SF Chapters in the 3D Volume.)
	31:0	**Clipped Primitives Output Count LDW** Total number of primitives output by the clipper stage. This count is updated for every primitive output by the clipper stage, as long as Statistics Enable is set in SF_STATE (see the Clipper and SF Chapters in the 3D Volume.)

Clock Gating Messages

CGMSG - Clock Gating Messages

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	08104h

Clock Gating Messages Register

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:10	**Reserved** Access: RO Reserved
	9	**Media sampler Clock gating control message** Access: R/W Gate Media sampler Clock Message : '0' : Clock Un-gate Request (un-gates the scmsclk clock) '1' : Clock Gate Request (gates the scmsclk clock)
	8	**SFC 1 Clock gating control message** Access: R/W Gate SFC 1 (2nd Vbox) Clock gate Message : '0' : SFC 1 Clock Un-gate Request (un-gates the cmclk clock in the 2nd Media block) '1' : SFC 1 Clock Gate Request (gates the cmclk clockin the 2nd Media block)
	7	**SFC 0 Clock gating control message** Access: R/W Gate SFC 0 (1st Vbox) Clock gate Message : '0' : SFC 0 Clock Un-gate Request (un-gates the cmclk clock in the 1st Media block) '1' : SFC 0 Clock Gate Request (gates the cmclk clockin the 1st Media block)

CGMSG - Clock Gating Messages

	6	**Media 1 Clock gating control message** Access: R/W Gate Media 1 (2nd Vbox) Clock Message : '0' : Media 1 Clock Un-gate Request (un-gates the cmclk clock in the 2n Media block) '1' : Media 1 Clock Gate Request (gates the cmclk clockin the 2nd Media block)
	5	**Reserved**
	4	**Reserved**
	3	**Fix Function Clock gating Control Message** Access: R/W Gate Fix Clock Message : '0' : Fix Clock Un-gate Request (un-gates the cfclk/cf2xclk clock) '1' : Fix Clock Gate Request (gates the cfclk/cf2xclk clock)
	2	**VEbox Clock gating Control message** Access: R/W Gate VE-box Clock Message : '0' : VEbox Clock Un-gate Request (un-gates the cvclk clock) '1' : VEbox Clock Gate Request (gates the cvclk clock)
	1	**Media 0 Clock Gating Control Message** Access: R/W Gate Media Clock Message : '0' : Media 0 Clock Un-gate Request (un-gates the cmclk clock) '1' : Media 0 Clock Gate Request (gates the cmclk clock)
	0	**Row Clock Gating Control Message** Access: R/W Gate Row Clocks Message : '0' : Row Clock Un-gate Request (un-gates the crclk and cr2xclk clocks) '1' : Row Clock Gate Request (gates the crclk and cr2xclk clocks)

Color/Depth Write FIFO Watermarks

CZWMRK - Color/Depth Write FIFO Watermarks

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04060h

This register is directly mapped to the current Virtual Addresses in the MTTLB (Texture and constant cache TLB).

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:18	**Color Wr Burst Size** This is the maximum size of the requests burst, from the last High Watermark trip, before reevaluating the High Watermark again.
	17:16	**Reserved** Format: MBZ
	15:12	**Color Wr FIFO High Watermark** This is the number of accumulated Color writes that will trigger a Burst of Z Writes.
	11:6	**Z Wr Burst Size** This is the maximum size of the requests burst, from the last High Watermark trip, before reevaluating the High Watermark again.
	5:4	**Reserved** Format: MBZ
	3:0	**Z Wr FIFO High Watermark** This is the number of accumulated Depth writes that will trigger a Burst of Z Writes.

Config to MCI HI

CFGTOMCIDFTHI - Config to MCI HI			
Register Space:	MMIO: 0/2/0		
Source:	BSpec		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	098A0h		
Config to MCI and DFT Ring			
DWord	**Bit**	**Description**	
0	31	**CFG to MCI HI dispatch**	
		Access:	R/WC
	30	**CFG to MCI HI error clear**	
		Access:	R/WC
	29:20	**RSVD_29_20**	
		Access:	R/WC
	19:0	**MCI DFT Ring Write data**	
		Access:	R/WC

Config to MCI LO

CFGTOMCIDFTLO - Config to MCI LO		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0989Ch	
Config to MCI and DFT Ring		
DWord	**Bit**	**Description**
0	31	**CFG to MCI LO dispatch** Access: R/WC
	30:0	**MCI DFT Ring Write data** Access: R/WC

Config to MCI STATUS1

CFGTOMCIDFTSTATUS1 - Config to MCI STATUS1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	098A4h	
Config to MCI and DFT Ring		
DWord	**Bit**	**Description**
0	31:5	**RSVD_31_5** Access: RO
	4	**Report fifo empty** Access: RO
	3	**Reserved**
	2	**Reserved**
	1	**mci fifo overflow** Access: RO
	0	**Report fifo overflow** Access: RO

Configuration Register0 for RPMunit

CONFIG0 - Configuration Register0 for RPMunit

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00D00h

Lock bit LOCK applies to all RW/L fields in this register. Lock is overridden during context restore.

DWord	Bit	Description
0	31	**Lock for RW/L Fields in this Register** Access: R/W Lock 0 = Bits of CONFIG0 register are R/W. 1 = All bits of CONFIG0 register are RO (including this lock bit). Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). Lock is reset on a restore after context is captured.
	30	**Engineering Sample/Pre-Production part Indicator** Access: R/W Lock Bit30 will be written by BIOS to indicate production/ES sample indication - polarity matches the CPU MSR SR 0xce[27] 1'b0 - production part. 1'b1 - engineering sample/pre-production part.
	29:5	**Placeholder Bits Config0** Access: R/W Lock Placeholder bits for implementation or ECO loops.
	4	**Prevent PowerGate Disable Config Bit** Access: R/W Lock Prevents Powergate License disabling config programming 0xD30[1] 'b0 - PowerGate License handshake enabled /disabled as per 0xD30[1] (default) 'b1 - PowerGate License disable config programming with bit 0xD30[0] is disabled. PowerGate license handshake is always enabled.
	3	**Reserved**
	2	**Reserved**
	1	**Reserved**
	0	**Disable TSC Synchronization** Access: R/W Lock 'b0 - TSC synchronization enabled in GT (default) 'b1 - TSC synchronization DISABLED in GT

Configuration Register1 for RPMunit

CONFIG1 - Configuration Register1 for RPMunit		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00D04h	
Lock bit LOCK applies to all RW/L fields in this register. Lock is overridden during context restore.		
DWord	**Bit**	**Description**
0	31	**Lock for RW/L Fields in this Register** Access: R/W Lock 0 = Bits of CONFIG0 register are R/W. 1 = All bits of CONFIG0 register are RO (including this lock bit). Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). Lock is reset on a restore after context is captured.
	30:9	**Placeholder Bits BitField** Access: R/W Lock Placeholder bits for implementation or ECO loops.
	8:0	**Placeholder Bits L3FREQTHRESH** Access: R/W Lock Placeholder bits for implementation or ECO loops.

Configuration Register for RCPunit

RCPCONFIG - Configuration Register for RCPunit

Register Space:	MMIO: 0/2/0
Default Value:	0x0000000F
Size (in bits):	32
Address:	00D08h

Unit Level Clock Gating Control Registers

DWord	Bit	Description
0	31:5	**Placeholder Bits Config0 ECO Loops** Access: R/W Lock Placeholder bits for implementation or ECO loops.
	4	**Reserved**
	3	**RPMunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).
	2	**MGSRunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).
	1	**MDRBunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).

RCPCONFIG - Configuration Register for RCPunit

	0	**MCRunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).

Context Load Protocol Register BLT

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000 [KBL]	
Size (in bits):	32	
Address:	04014h	
DWord	**Bit**	**Description**
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - BCS 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - BCS 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - BCS 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - BCS 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - BCS 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT

	10	**Context Load Protocol Register - BCS 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - BCS 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - BCS 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - BCS 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - BCS 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - BCS 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - BCS 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT

	3	**Context Load Protocol Register - BCS 3** Default Value: 0b Access: R/W Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - BCS 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 2 = Request from BCS to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - BCS 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - BCS 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register CS

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000 [KBL]
Size (in bits):	32
Address:	04004h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - CS 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - CS 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - CS 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - CS 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - CS 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS

	10	**Context Load Protocol Register - CS 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - CS 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - CS 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - CS 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - CS 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - CS 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - CS 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS

	3	**Context Load Protocol Register - CS 3** Default Value: 0b Access: R/W Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - CS 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 2 = Request from CS to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - CS 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - CS 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register VCS0

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000 [KBL]	
Size (in bits):	32	
Address:	04008h	

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VCS0 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VCS0 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VCS0 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VCS0 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VCS0 11** Default Value: 0b Access: R/W For Future Use.

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0

		This bit is self clear.
	10	**Context Load Protocol Register - VCS0 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VCS0 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VCS0 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VCS0 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VCS0 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VCS0 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VCS0 4**

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0

	Default Value:	0b
	Access:	R/W

For Future Use.
This bit is self clear.

3 — Context Load Protocol Register - VCS0 3

Default Value:	0b
Access:	R/W

Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete.
This bit is self clear.

2 — Context Load Protocol Register - VCS0 2

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS0)
Bit 2 = Request from VCS0 to GAM for context save readiness. GAM will acknowledge appropriated steps taken.
This bit is self clear.

1 — Context Load Protocol Register - VCS0 1

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS0)
Bit 1 = Context Launched.
This bit is self clear.

0 — Context Load Protocol Register - VCS0 0

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS0)
Bit 0 = Context Available.
This bit is self clear.

Context Load Protocol Register VCS1

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000 [KBL]
Size (in bits):	32
Address:	0400Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VCS1 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VCS1 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VCS1 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VCS1 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VCS1 11** Default Value: 0b Access: R/W For Future Use.

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

		This bit is self clear.
	10	**Context Load Protocol Register - VCS1 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VCS1 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VCS1 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VCS1 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VCS1 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VCS1 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VCS1 4**

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

	Default Value:	0b
	Access:	R/W

For Future Use.
This bit is self clear.

3 — **Context Load Protocol Register - VCS1 3**

Default Value:	0b
Access:	R/W

Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete.
This bit is self clear.

2 — **Context Load Protocol Register - VCS1 2**

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS1)
Bit 2 = Request from VCS1 to GAM for context save readiness. GAM will acknowledge appropriated steps taken.
This bit is self clear.

1 — **Context Load Protocol Register - VCS1 1**

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS1)
Bit 1 = Context Launched
This bit is self clear.

0 — **Context Load Protocol Register - VCS1 0**

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VCS1)
Bit 0 = Context Available.
This bit is self clear.

Context Load Protocol Register VEBX

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000 [KBL]	
Size (in bits):	32	
Address:	04010h	

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VEBX 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VEBX 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VEBX 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VEBX 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VEBX 11** Default Value: 0b Access: R/W For Future Use.

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX

		This bit is self clear.
	10	**Context Load Protocol Register - VEBX 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VEBX 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VEBX 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VEBX 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VEBX 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VEBX 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VEBX 4**

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX

	Default Value:	0b
	Access:	R/W

For Future Use.
This bit is self clear.

3 — Context Load Protocol Register - VEBX 3

Default Value:	0b
Access:	R/W

Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete.
This bit is self clear.

2 — Context Load Protocol Register - VEBX 2

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VEBX)
Bit 2 = Request from VEBX to GAM for context save readiness. GAM will acknowledge appropriated steps taken.
This bit is self clear.

1 — Context Load Protocol Register - VEBX 1

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VEBX)
Bit 1 = Context Launched.
This bit is self clear.

0 — Context Load Protocol Register - VEBX 0

Default Value:	0b
Access:	R/W

Context Load Protocol Register
(Written by VEBX)
Bit 0 = Context Available.
This bit is self clear.

Context Restore Request To TDL

TDL_CONTEXT_RESTORE - Context Restore Request To TDL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E440h

DWord	Bit	Description
0	31:17	**Reserved** Format: MBZ
	16	**Context Restore Mask** Value: 1 — Name: — Description: Bit 0 and bit 16 both need to be 1 for Context restore request
	15:1	**Reserved** Format: MBZ
	0	**Context Restore** Value: 1 — Name: — Description: Bit 0 and bit 16 both need to be 1 for Context restore request

Context Save Request To TDL

TDL_CONTEXT_SAVE - Context Save Request To TDL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4FCh

DWord	Bit	Description
0	31:17	**Reserved** Format: MBZ
	16	**Context Save Mask** Value: 1 — Name: — Description: Bit 0 and Bit 16 both need to be '1' for Context Save Request
	15:1	**Reserved** Format: MBZ
	0	**Context Save** Value: 1 — Name: — Description: Bit 0 and Bit 16 both need to be '1' for Context Save Request

Context Sizes

CXT_SIZE - Context Sizes

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	021A8h

The actual size of a logical rendering context is the amount of data stored/restored during a context switch and is measured in 64B cache lines.

This register will be power context save/restored. Note that this register will default to the correct value, so software should not have to modify it.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	7:0	**Reserved** Format: MBZ

Context Status1 for RCS-BE

CS_CONTEXT_STATUS1 - Context Status1 for RCS-BE

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1
Address:	02184h

This register is for maintaining HW internal state of RCS-BE per context. This register is context save/restore per context. This register should not be written by SW.

DWord	Bit	Description
0	31:16	**Mask Bits** Access: WO Format: Mask Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:14	**Reserved** Format: PBC
	13:8	**Reserved** Format: PBC
	7	**Preempted Batch Buffer RS Control Stop Flag** Format: Flag This field specifies RS Control Stop Flag when a batch buffer is preempted. This is for HW internal use and should not be written by SW. This bit gets reset when RS_PREEMPTED field of RS_PREEMPT_STATUS is written Zero. This bit is set by: • Ctx restore of this bit • MI_RS_CONTROL_STOP (except for the ctx restore command) This bit is cleared by: • MI_RS_CONTROL_START • Any Batch start except resubmitted RS batch • A batch end that doesn't include preemption • Ctx save Writing 0 to bit[0] of the RS STATUS register

CS_CONTEXT_STATUS1 - Context Status1 for RCS-BE

	6	**Pending Indirect State Dirty Bit** Format: U1 This field keeps track of whether or not an indirect state pointer command has been parsed in the current context. Clears either on a context save or explicitly through a flush command.
	5:0	**Pending Indirect State Counter** Access: RO Format: U6 This field keeps track of the maximum number of indirect state pointers pending in the system. When the register is saved/restored, it saves either a value of 1 or 0.

Context Status Buffer Contents

CTXT_ST_BUF - Context Status Buffer Contents

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	R/W
Size (in bits):	384
Trusted Type:	1
Address:	02370h-0239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_RCSUNIT
Address:	12370h-1239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VCSUNIT0
Address:	1A370h-1A39Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VECSUNIT
Address:	1C370h-1C39Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VCSUNIT1
Address:	22370h-2239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_BCSUNIT

Contents of the Execlist 0 in HW.

Programming Notes

This structure contains the Context Switch status locations Context Status 0 to Context Status 5.

DWord	Bit	Description
0	63:32	**Context Status 0 UDW** Format: Context Status
	31:0	**Context Status 0 LDW** Format: Context Status
1	63:32	**Context Status 1 UDW** Format: Context Status

CTXT_ST_BUF - Context Status Buffer Contents		
	31:0	**Context Status 1 LDW** Format: **Context Status**
2	63:32	**Context Status 2 UDW** Format: **Context Status**
	31:0	**Context Status 2 LDW** Format: **Context Status**
3	63:32	**Context Status 3 UDW** Format: **Context Status**
	31:0	**Context Status 3 LDW** Format: **Context Status**
4	63:32	**Context Status 4 UDW** Format: **Context Status**
	31:0	**Context Status 4 LDW** Format: **Context Status**
5	63:32	**Context Status 5 UDW** Format: **Context Status**
	31:0	**Context Status 5 LDW** Format: **Context Status**

Context Timestamp Count

CTX_TIMESTAMP - Context Timestamp Count

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	023A8h-023ABh
Name:	Context Timestamp Count
ShortName:	CTX_TIMESTAMP_RCSUNIT

Address:	123A8h-123ABh
Name:	Context Timestamp Count
ShortName:	CTX_TIMESTAMP_VCSUNIT0

Address:	1A3A8h-1A3ABh
Name:	Context Timestamp Count
ShortName:	CTX_TIMESTAMP_VECSUNIT

Address:	1C3A8h-1C3ABh
Name:	Context Timestamp Count
ShortName:	CTX_TIMESTAMP_VCSUNIT1

Address:	223A8h-223ABh
Name:	Context Timestamp Count
ShortName:	CTX_TIMESTAMP_BCSUNIT

This register provides a mechanism to obtain cumulative run time of a GPU context on HW. This register gets context save/restored on a context switch. SW must reset this register on very first submission of a context to HW, then afterwards gets context save/restored maintaining the cumulative run time of the corresponding context.
This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register gets reset on an engine reset.

This register is context save restore on a context switch.

DWord	Bit	Description
0	31:0	**Timestamp Value** Format: U32 The granularity of this toggle is at the rate of the bit 3 in the "Reported Timestamp Count" register(0x2358).. The toggle will be 8 times slower that "Reported Timestamp Count". The granularity of the time stamp base unit for "Reported Timestamp Count" is defined in the “Timestamp Bases[SKL+]” subsection in Power Management chapter.

Control Register for Fault and Halt

FH_MODE - Control Register for Fault and Halt

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	042A4h

This register is used to control the different fault and halt modes.

DWord	Bit	Description
0	31	**Disable Blocking Page Fault** Default Value: 0b Access: R/W When disabled h/w would not set the "blocking fault" bit in the streaming page fault descriptor for the fault and halt generated page faults. FH_MODE: 0: Enable "blocking page fault" indicator for Fault and Halt 1: Disable "blocking page fault" indicator for Fault and Halt
	30	**Enable Forward Progress under F and H based page faults - Render engine only** Default Value: 0b Access: R/W Once set, the page walker will not wait for accesses that are hitting page faults on fault and halt cases, and marking accesses as "invalid" making forward progress. Such case may corrupt the frame or may require TDR if surface is CRITICAL. Invalid accesses will return garbage content. GFX Driver can set this bit in the middle of an active context, h/w should clear the bit when an active context completes. Usage model will be as driver hits a fault and halt and interrupts the driver, driver will set this bit is it needs forward progress. The behavior should only be applicable to the running context. FWDPROG: 0: Forwards progress under fault and halt is only possible with page response. 1: Enable forward progress under falt and halt w/o the need of paging services.
	29	**Enable Interrupt Generation** Default Value: 0b Access: R/W Enable interrupt generation on fault and halt page when resume mode is enabled: An interrupt can be generated on page fault with the fault and halt mode when hardware is programmed to resume(rather than wait/halt). The generation of interrupt needs to be explicitly enabled via this register bit. ENINTR: 0: No interrupt is generated on fault and halt page fault.

FH_MODE - Control Register for Fault and Halt

		1: Enable interrupt generation for fault and halt based page faults when h/w is programmed to resume.
	28:0	**Reserved FH_MODE Bits 28** Default Value: 00000000000000000000000000000b Access: R/W Future Use.

Count Active Channels Dispatched

TS_GPGPU_THREADS_DISPATCHED - Count Active Channels Dispatched

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02290h

This register is used to count the number of active channels that TS sends for dispatch. For each dispatch the active bits in the execution mask are summed and added to this register. This register is reset when a write occurs to 2290h

DWord	Bit	Description
0	63:32	**GPGPU_THREADS_DISPATCHED UDW** Format: U32 This count is increased by the number of active bits in the execution mask each time the TS sends a GPGPU dispatch.
	31:0	**GPGPU_THREADS_DISPATCHED LDW** Format: U32 This count is increased by the number of active bits in the execution mask each time the TS sends a GPGPU dispatch.

CSC_COEFF

CSC_COEFF	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	Double Buffered
Size (in bits):	192
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_MODE
Address:	49010h-49027h
Name:	Pipe CSC Coefficients
ShortName:	CSC_COEFF_A
Power:	PG1
Reset:	soft
Address:	49110h-49127h
Name:	Pipe CSC Coefficients
ShortName:	CSC_COEFF_B
Power:	PG2
Reset:	soft
Address:	49210h-49227h
Name:	Pipe CSC Coefficients
ShortName:	CSC_COEFF_C
Power:	PG2
Reset:	soft

DWord	Bit	Description
0	31:16	**RY** Format: CSC COEFFICIENT FORMAT
	15:0	**GY** Format: CSC COEFFICIENT FORMAT
1	31:16	**BY** Format: CSC COEFFICIENT FORMAT
	15:0	**Reserved** Format: MBZ
2	31:16	**RU**

CSC_COEFF		
		Format: CSC COEFFICIENT FORMAT
	15:0	**GU** Format: CSC COEFFICIENT FORMAT
3	31:16	**BU** Format: CSC COEFFICIENT FORMAT
	15:0	**Reserved** Format: MBZ
4	31:16	**RV** Format: CSC COEFFICIENT FORMAT
	15:0	**GV** Format: CSC COEFFICIENT FORMAT
5	31:16	**BV** Format: CSC COEFFICIENT FORMAT
	15:0	**Reserved** Format: MBZ

CSC_MODE

CSC_MODE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank
Address:	49028h-4902Bh
Name:	Pipe CSC Mode
ShortName:	CSC_MODE_A
Power:	PG1
Reset:	soft
Address:	49128h-4912Bh
Name:	Pipe CSC Mode
ShortName:	CSC_MODE_B
Power:	PG2
Reset:	soft
Address:	49228h-4922Bh
Name:	Pipe CSC Mode
ShortName:	CSC_MODE_C
Power:	PG2
Reset:	soft

Description
Writes to this register arm CSC registers for this pipe.

DWord	Bit	Description
0	31:2	**Reserved** Format: MBZ
	1	**CSC Position** Selects the CSC position in the pipe. This is ignored when split gamma mode is selected in the pipe config register.

Value	Name	Description
0b	CSC After	CSC is after gamma
1b	CSC Before	CSC is before gamma

CSC_MODE		
	0	**Reserved** Format: MBZ

CSC_POSTOFF

CSC_POSTOFF	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000
Access:	Double Buffered
Size (in bits):	96
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_MODE
Address:	49040h-4904Bh
Name:	Pipe CSC Post-Offsets
ShortName:	CSC_POSTOFF_A
Power:	PG1
Reset:	soft
Address:	49140h-4914Bh
Name:	Pipe CSC Post-Offsets
ShortName:	CSC_POSTOFF_B
Power:	PG2
Reset:	soft
Address:	49240h-4924Bh
Name:	Pipe CSC Post-Offsets
ShortName:	CSC_POSTOFF_C
Power:	PG2
Reset:	soft

The post-offset is intended to add an offset from 0 on the Y or RGB channels and to convert UV channels from 2's complement to excess 0.5 as they exit pipe color space conversion (CSC).

DWord	Bit	Description
0	31:13	**Reserved** Format: MBZ
	12:0	**PostCSC High Offset** This value is used to give an offset to the high color channel as it exits CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).
1	31:13	**Reserved** Format: MBZ

CSC_POSTOFF		
	12:0	**PostCSC Medium Offset** This value is used to give an offset to the medium color channel as it exits CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).
2	31:13	**Reserved** Format: MBZ
	12:0	**PostCSC Low Offset** This value is used to give an offset to the low color channel as it exits CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).

CSC_PREOFF

CSC_PREOFF	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000
Access:	Double Buffered
Size (in bits):	96
Double Buffer Update Point:	Start of vertical blank after armed
Double Buffer Armed By:	Write to CSC_MODE
Address:	49030h-4903Bh
Name:	Pipe CSC Pre-Offsets
ShortName:	CSC_PREOFF_A
Power:	PG1
Reset:	soft
Address:	49130h-4913Bh
Name:	Pipe CSC Pre-Offsets
ShortName:	CSC_PREOFF_B
Power:	PG2
Reset:	soft
Address:	49230h-4923Bh
Name:	Pipe CSC Pre-Offsets
ShortName:	CSC_PREOFF_C
Power:	PG2
Reset:	soft

The pre-offset is intended to remove an offset from 0 on the Y or RGB channels and to convert UV channels from excess 0.5 to 2's complement as they enter pipe color space conversion (CSC).

DWord	Bit	Description
0	31:13	**Reserved** Format: MBZ
	12:0	**PreCSC High Offset** This value is used to give an offset to the high color channel as it enters CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).
1	31:13	**Reserved** Format: MBZ

CSC_PREOFF		
	12:0	**PreCSC Medium Offset** This value is used to give an offset to the medium color channel as it enters CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).
2	31:13	**Reserved** Format: MBZ
	12:0	**PreCSC Low Offset** This value is used to give an offset to the low color channel as it enters CSC logic. The value is a 2's complement fraction allowing offsets between -1 and +1 (exclusive).

CSPREEMPT

CSPREEMPT - CSPREEMPT

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	024B0h
Name:	CSPREEMPT
ShortName:	CSPREEMPT
Address:	224B0h
Name:	BCSPREEMPT
ShortName:	BCSPREEMPT
Address:	124B0h
Name:	VCSPREEMPT
ShortName:	VCSPREEMPT
Address:	1A4B0h
Name:	VECSPREEMPT
ShortName:	VECSPREEMPT

Programming Notes

This is for HW internal usage and must not be written by SW.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:1	**Reserved** Format: MBZ
	0	**Unnamed** Format: Disable This is a message bit written by the cross CS in case of GT4-CBR/SFR mode of operation. To set this bit both bit[0] and bit[16] (mask) needs to be set. This bit set indicates CS in other GT has reached a preemption point. This bit gets reset by CS when preemption takes place.

CTX REG 1

CTXREG1 - CTX REG 1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Size (in bits):	32	
Address:	00FF4h-00FF7h	
DWord	**Bit**	**Description**
	31:0	**CTXSIZE** Default Value: 0000026Fh Access: RO Register to store value for number of CTX DWORD.

CTX reg 2

CTXREG2 - CTX reg 2		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00FFCh-00FFFh	
DWord	**Bit**	**Description**
0	31:1	**CTX Register 2** Access: R/W RSVD
	0	**CTXRESTOREDONE** Access: R/W CTX restore done bit. Will be written to 1 during the last CTX restore cycle.

CUR_BASE

CUR_BASE

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe not enabled

Address:	70084h-70087h
Name:	Cursor Base Address
ShortName:	CUR_BASE_A
Power:	PG1
Reset:	soft

Address:	71084h-71087h
Name:	Cursor Base Address
ShortName:	CUR_BASE_B
Power:	PG2
Reset:	soft

Address:	72084h-72087h
Name:	Cursor Base Address
ShortName:	CUR_BASE_C
Power:	PG2
Reset:	soft

Writes to this register arm cursor registers for this pipe.

DWord	Bit	Description
0	31:12	**Cursor Base 31 12** Format: GraphicsAddress[31:12] This field specifies bits 31:12 of the graphics address of the base of the cursor for hi-res mode. When performing 180 degree rotation, this address does not need to change, hardware will internally offset to start from the last pixel of the last line of the cursor. **Workaround** Workaround : To prevent false VT-d type 6 errors, use 64KB address alignment and allocate an extra 2 Page Table Entries (PTEs) beyond the end of the displayed surface. Only the PTEs will be used, not the pages themselves. **Restriction**

CUR_BASE		
		Restriction : The cursor surface address must be 4K byte aligned. The cursor must be in linear memory, it cannot be tiled.
	11:7	**Reserved**
	6:4	**Reserved**
	3	**Reserved**
	2	**Reserved**
	1:0	**Reserved**

CUR_CTL

CUR_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe not enabled; after armed
Double Buffer Armed By:	Write to CUR_BASE or cursor not enabled

Address:	70080h-70083h
Name:	Cursor Control
ShortName:	CUR_CTL_A
Power:	PG1
Reset:	soft

Address:	71080h-71083h
Name:	Cursor Control
ShortName:	CUR_CTL_B
Power:	PG2
Reset:	soft

Address:	72080h-72083h
Name:	Cursor Control
ShortName:	CUR_CTL_C
Power:	PG2
Reset:	soft

The cursor is enabled by programming a valid cursor mode in the cursor mode select fields.
The cursor is disabled by programming all 0s in the cursor mode select fields.

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Reserved**
	26	**Gamma Enable** This bit enables pipe gamma correction for the cursor pixel data. In VGA pop-up operation, the cursor data will always bypass gamma. Value: 0b — Name: Disable

CUR_CTL

	1b	Enable

25 Reserved

24 Pipe CSC Enable

This bit enables pipe color space conversion for the cursor pixel data.

Value	Name
0b	Disable
1b	Enable

23 Allow Double Buffer Update Disable

Access:	R/W

This field controls whether double buffer updates are allowed to be disabled for this cursor. The DOUBLE_BUFFER_CTL register can be configured to globally disable double buffer updates for resources that allow them to be disabled.

Value	Name
0b	Not Allowed
1b	Allowed

22:16 Reserved

Format:	MBZ

15 180 Rotation

This mode causes the cursor image to be rotated 180 degrees.
In addition to setting this bit, the cursor position must be adjusted to match the physical orientation of the display.

Value	Name
0b	No rotation
1b	180 degree rotation

Restriction
Restriction : Only 32 bits per pixel cursors can be rotated. This field must be zero when the cursor format is 2 bits per pixel.

14 Trickle Feed Enable

Value	Name
0b	Enable
1b	Disable

Restriction
Restriction : Do not program this field to 1b.

13:12 Reserved

11:10 Force Alpha Plane Select

This field selects which planes the cursor alpha value will be forced for. It is used together with

CUR_CTL

the Force Alpha Value field.

Value	Name	Description
00b	Disable	Disable alpha forcing
01b	Pipe CSC Enabled	Enable alpha forcing where cursor overlaps a plane that has enabled pipe CSC
10b	Pipe CSC Disabled	Enable alpha forcing where cursor overlaps plane that has disabled pipe CSC
11b	Reserved	Reserved

9:8 **Force Alpha Value**

This field controls the behavior of cursor when alpha blending onto certain plane pixels.
It is used together with the Force Alpha Plane Select field.

Value	Name	Description
00b	Disable	Cursor pixels alpha blend normally over any plane.
01b	50	Cursor pixels with alpha >= 50% are made fully opaque where they overlap the selected plane(s). Cursor pixels with alpha < 50% are made fully transparent where they overlap the selected plane(s).
10b	75	Cursor pixels with alpha >= 75% are made fully opaque where they overlap the selected plane(s). Cursor pixels with alpha < 75% are made fully transparent where they overlap the selected plane(s).
11b	100	Cursor pixels with alpha = 100% are made fully opaque where they overlap the selected plane(s). Cursor pixels with alpha < 100% are made fully transparent where they overlap the selected plane(s).

Restriction
Restriction : Force Alpha is only for use with ARGB cursor formats.

7:6 **Reserved**

5:0 **Cursor Mode Select**

This field selects the cursor mode.
Cursor is disabled when the selection is 000000b and enabled when the selection is any other value.
The cursor vertical size can be overriden by the size reduction mode.

Value	Name	Description
000000b	Disable	Cursor is disabled
000010b	128x128 32bpp AND/INV	128x128 32bpp AND/INVERT
000011b	256x256 32bpp AND/INV	256x256 32bpp AND/INVERT
000100b	64x64 2bpp 3-color	64x64 2bpp Indexed 3-color and transparency
000101b	64x64 2bpp 2-color	64x64 2bpp Indexed AND/XOR 2-color

CUR_CTL

000110b	64x64 2bpp 4-color	64x64 2bpp Indexed 4-color
000111b	64x64 32bpp AND/INV	64x64 32bpp AND/INVERT
100010b	128x128 32bpp ARGB	128x128 32bpp ARGB (8:8:8:8 MSB-A:R:G:B)
100011b	256x256 32bpp ARGB	256x256 32bpp ARGB (8:8:8:8 MSB-A:R:G:B)
100100b	64x64 32bpp AND/XOR	64x64 32bpp AND/XOR
100101b	128x128 32bpp AND/XOR	128x128 32bpp AND/XOR
100110b	256x256 32bpp AND/XOR	256x256 32bpp AND/XOR
100111b	64x64 32bpp ARGB	64x64 32bpp ARGB (8:8:8:8 MSB-A:R:G:B)
Others	Reserved	Reserved

Programming Notes

INVERT, XOR, and alpha blends may not look as expected when the plane underlying the cursor is YUV or extended range RGB.
Out of range RGB values will be clamped prior to alpha blending, INVERT, or XOR with cursor.
It is recommended to use Force Alpha when cursor is alpha blending onto an plane of a different color space or extended gamut.

The AND/INVERT format uses the most significant byte (MSB) to control the color.
If MSB is 0xFF: Cursor is opaque. Show cursor color from three least significant bytes.
If MSB is 0x00: Cursor is transparent. Three least significant bytes must be zero.
If MSB is not 0x00 or 0xFF: Cursor inverts the color of the surface underneath.

The AND/XOR format uses the most significant byte (MSB) to control the color.
If MSB is 0xFF: Cursor is opaque. Show cursor color from three least significant bytes.
If MSB is 0x00: Cursor is transparent. Three least significant bytes must be zero.
If MSB is not 0x00 or 0xFF: The three least significant bytes are XOR'd with the color of the surface underneath.

CUR_FBC_CTL

CUR_FBC_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe not enabled; after armed
Double Buffer Armed By:	Write to CUR_BASE or cursor not enabled

Address:	700A0h-700A3h
Name:	Cursor FBC Control
ShortName:	CUR_FBC_CTL_A
Power:	PG1
Reset:	soft

Address:	710A0h-710A3h
Name:	Cursor FBC Control
ShortName:	CUR_FBC_CTL_B
Power:	PG2
Reset:	soft

Address:	720A0h-720A3h
Name:	Cursor FBC Control
ShortName:	CUR_FBC_CTL_C
Power:	PG2
Reset:	soft

DWord	Bit	Description
0	31	**Size Reduction Enable** This enables cursor size reduction logic. The cursor engine will fetch and display the programmed reduced number of lines, then go transparent for the rest of the frame. **Value** / **Name** 0b — Disable 1b — Enable **Restriction**

CUR_FBC_CTL		
		Restriction : Cursor size reduction is not allowed with 2bpp cursor formats or cursor 180 degree rotation. The reduced scan lines field must be programmed with a valid value when cursor size reduction is enabled.
	30:8	**Reserved**
	7:0	**Reduced Scan Lines** This specifies the number of scan lines of cursor data to fetch and display when cursor size reduction is enabled. The value programmed is the size minus one. **Restriction** Restriction : The minimum size is 8 lines, programmed as 07h. The maximum size can not be greater than the normal size when size reduction is not enabled.

CUR_PAL

CUR_PAL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe not enabled
Address:	70090h-7009Fh
Name:	Cursor A Palette
ShortName:	CUR_PAL_A_*
Power:	PG1
Reset:	soft
Address:	71090h-7109Fh
Name:	Cursor B Palette
ShortName:	CUR_PAL_B_*
Power:	PG2
Reset:	soft
Address:	72090h-7209Fh
Name:	Cursor C Palette
ShortName:	CUR_PAL_C_*
Power:	PG2
Reset:	soft

The cursor palette provides color information when using the indexed cursor modes.
There are 4 instances of this register format per cursor.
The table below describes how the cursor mode and index value will select between the cursor palette colors, AND/XOR, transparency, and destination invert.

Index Value	2 color mode	3 color mode	4 color mode
00	CUR_PAL 0	CUR_PAL 0	CUR_PAL 0
01	CUR_PAL 1	CUR_PAL 1	CUR_PAL 1
10	Transparent	Transparent	CUR_PAL 2
11	Invert Destination	CUR_PAL 3	CUR_PAL 3

DWord	Bit	Description
0	31:24	**Reserved**

CUR_PAL		
	23:16	**Palette Red** This field is the cursor palette red value
	15:8	**Palette Green** This field is the cursor palette green value.
	7:0	**Palette Blue** This field is the cursor palette blue value.

CUR_POS

CUR_POS

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank or pipe not enabled

Address:	70088h-7008Bh
Name:	Cursor Position
ShortName:	CUR_POS_A
Power:	PG1
Reset:	soft

Address:	71088h-7108Bh
Name:	Cursor Position
ShortName:	CUR_POS_B
Power:	PG2
Reset:	soft

Address:	72088h-7208Bh
Name:	Cursor Position
ShortName:	CUR_POS_C
Power:	PG2
Reset:	soft

This register specifies the screen position of the cursor.
The origin of the cursor position is always the upper left corner of the display pipe source image area.
When performing 180 degree rotation, the cursor image is rotated by hardware, but the position is not, so it should be adjusted if it is desired to maintain the same apparent position on a physically rotated display.

Restriction
Restriction : The cursor must have at least a single pixel positioned over the pipe source area.

DWord	Bit	Description
0	31	**Y Position Sign** This specifies the sign of the vertical position of the cursor upper left corner.
	30:28	**Reserved** Format: MBZ

CUR_POS		
	27:16	**Y Position Magnitude** This specifies the magnitude of the vertical position of the cursor upper left corner in lines.
	15	**X Position Sign** This specifies the sign of the horizontal position of the cursor upper left corner.
	14:13	**Reserved** Format: MBZ
	12:0	**X Position Magnitude** This specifies the magnitude of the horizontal position of the cursor upper left corner in pixels.

CUR_SURFLIVE

CUR_SURFLIVE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	700ACh-700AFh
Name:	Cursor Live Base Address
ShortName:	CUR_SURFLIVE_A
Power:	PG1
Reset:	soft
Address:	710ACh-710AFh
Name:	Cursor Live Base Address
ShortName:	CUR_SURFLIVE_B
Power:	PG2
Reset:	soft
Address:	720ACh-720AFh
Name:	Cursor Live Base Address
ShortName:	CUR_SURFLIVE_C
Power:	PG2
Reset:	soft

There is one instance of this register for each pipe.

DWord	Bit	Description
0	31:12	**Live Surface Base Address** This gives the live value of the surface base address as being currently used for the cursor.
	11:0	**Reserved** Format: MBZ

Current Context Register

CCID - Current Context Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02180h-02183h
Name:	Current Context Register
ShortName:	CCID_RCSUNIT

Address:	12180h-12183h
Name:	Current Context Register
ShortName:	CCID_VCSUNIT0

Address:	1A180h-1A183h
Name:	Current Context Register
ShortName:	CCID_VECSUNIT

Address:	1C180h-1C183h
Name:	Current Context Register
ShortName:	CCID_VCSUNIT1

Address:	22180h-22183h
Name:	Current Context Register
ShortName:	CCID_BCSUNIT

Description	Source
This register contains the 4 KB-aligned Graphics Memory Address to which the engine must save/restore its state during IDLE sequencing. This register should be programmed only in ringbuffer mode of scheduling.	
On SKL and its derivatives context address programmed should support a memory surface of size 4KB and with memory surface initialized to zeros (0x0).	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS

Programming Notes	Source
The CCID register must be written directly (via MMIO) or MI_LOAD_REGISTER_IMMEDIATE command as part of the initialization sequence of the command streamer in ring buffer mode of scheduling.	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
The CCID register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle). Note that, under normal conditions, the CCID register should only be updated from the command stream using the MI_SET_CONTEXT command.	RenderCS

DWord	Bit	Description

CCID - Current Context Register

0 | 31:12 | Context Address

Format:	GraphicsAddress[31:12]

This field contains the 4 KB-aligned Graphics Memory Address to which the engine must save/restore its state during IDLE sequencing.

11:10 | Reserved

Source:	RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

9 | HD DVD Context

Source:	RenderCS

Value	Name	Description
0h	Regular Context	
1h	HD DVD Context	Special considerations for TDP allow for higher voltage and frequency.

9 | Reserved

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

8 | Reserved8

Source:	RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	U1

7 | Reserved

6:4 | Reserved

Default Value:	0
Source:	RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS

3 | Extended State Save Enable

Source:	RenderCS
Format:	Enable

If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, is saved as part of switching away from this logical context.

3 | Reserved

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

CCID - Current Context Register

	2	**Extended State Restore Enable** Source: RenderCS Format: Enable If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, was loaded (or restored) as part of switching to this logical context.
	2	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	1	**Reserved** Source: RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	0	**Valid** Format: U1

Value	Name	Description
0h	Invalid **[Default]**	The other fields of this register are invalid.
1h	Valid	The other fields of this register are valid.

Customizable Event Creation 0-0

CEC0-0 - Customizable Event Creation 0-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02770h

This register is used to define custom counter event 0, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC0-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the last CEC block as the input bus to CEC0 block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC0-0 - Customizable Event Creation 0-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC0 comparator when comparing the compare value to the value active on the CEC0 conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 1-0

CEC1-0 - Customizable Event Creation 1-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02778h

This register is used to define custom counter event 1, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value / Name / Description / Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC1-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value / Name / Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC1-0 - Customizable Event Creation 1-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 1-1

CEC1-1 - Customizable Event Creation 1-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0277Ch

This register configures the input conditioning portion of CEC (custom event creation) block 1, bit definitions in this register refer to the CEC block diagram.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

DWord	Bit	Description
0	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

Customizable Event Creation 2-0

CEC2-0 - Customizable Event Creation 2-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02780h

This register is used to define custom counter event 2, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC2-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC2-0 - Customizable Event Creation 2-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 2-1

CEC2-1 - Customizable Event Creation 2-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02784h

This register configures the input conditioning portion of CEC (custom event creation) block 2, bit definitions in this register refer to the CEC block diagram.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

DWord	Bit	Description
	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

Customizable Event Creation 3-0

CEC3-0 - Customizable Event Creation 3-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02788h

This register is used to define custom counter event 3, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC3-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC3-0 - Customizable Event Creation 3-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 3-1

CEC3-1 - Customizable Event Creation 3-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0278Ch

This register configures the input conditioning portion of CEC (custom event creation) block 3, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

DWord	Bit	Description
	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

Customizable Event Creation 4-0

CEC4-0 - Customizable Event Creation 4-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02790h

This register is used to define custom counter event 4, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC4-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the last CEC block as the input bus to CEC0 block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC4-0 - Customizable Event Creation 4-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 5-0

CEC5-0 - Customizable Event Creation 5-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02798h

This register is used to define custom counter event 5, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC5-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC5-0 - Customizable Event Creation 5-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 5-1

CEC5-1 - Customizable Event Creation 5-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0279Ch

This register configures the input conditioning portion of CEC (custom event creation) block 5, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

DWord	Bit	Description
	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

Customizable Event Creation 6-0

CEC6-0 - Customizable Event Creation 6-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027A0h

This register is used to define custom counter event 6, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC6-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC6-0 - Customizable Event Creation 6-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 6-1

CEC6-1 - Customizable Event Creation 6-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027A4h

This register configures the input conditioning portion of CEC (custom event creation) block 6, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.
	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Bit 31:16 values:

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

Bit 15:0 values:

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

Customizable Event Creation 7-0

CEC7-0 - Customizable Event Creation 7-0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027A8h

This register is used to define custom counter event 7, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Programming Notes** 0b / Pass-through / Input bit is passed through to comparator as is / 1b / Negated / Input bit is negated before passing to comparator / Workaround : If the input bit is negated using any bit in this field, then the corresponding Consderations bit in the CEC7-1 register must also be set.
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC7-0 - Customizable Event Creation 7-0

2:0 **Compare Function**

Format:	U3

This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 7-1

CEC7-1 - Customizable Event Creation 7-1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027ACh

This register configures the input conditioning portion of CEC (custom event creation) block 3, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:16	**Considerations** This 16-bit field allows individual bits of the bus selected as the input to CEC block to be delayed by 1 clock relative to the non-delayed bits in the bus (see block diagram in the Custom Event Counters section). See CEC0-1 register definition for an example use case.

Value	Name	Description
0b	Live	Input bit is not delayed by 1 clock before event calculation
1b	Delayed	Input bit is delayed by 1 clock before event calculation

DWord	Bit	Description
	15:0	**Mask** This 16-bit field allows individual input bits to be ignored in custom event calculation. See block diagram in the Custom Event Counters section for more details on where this field is used.

Value	Name	Description
0b	Unmasked	Input bit is considered in event calculation
1b	Masked	Input bit is ignored in event calculation

CVS TLB LRA 0

CVS_TLB_LRA_0 - CVS TLB LRA 0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x3F201F00 [KBL]
Size (in bits):	32
Address:	04A20h

DWord	Bit	Description
0	31	**Reserved** Default Value: 0b Access: RO
	30:24	**CVS LRA1 Max** Default Value: 0111111b Access: R/W Maximum value of programmable LRA1.
	23	**Reserved** Default Value: 0b Access: RO
	22:16	**CVS LRA1 Min** Default Value: 0100000b Access: R/W Minimum value of programmable LRA1.
	15	**Reserved** Default Value: 0b Access: RO
	14:8	**CVS LRA0 Max** Default Value: 0011111b Access: R/W Maximum value of programmable LRA0.
	7	**Reserved** Default Value: 0b Access: RO

CVS_TLB_LRA_0 - CVS TLB LRA 0

	6:0	**CVS LRA0 Min** Default Value: 0000000b Access: R/W Minimum value of programmable LRA0.

CVS TLB LRA 1

CVS_TLB_LRA_1 - CVS TLB LRA 1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x7F717040
Size (in bits):	32
Address:	04A24h

DWord	Bit	Description
0	31	**Reserved** Default Value: 0b Access: RO
	30:24	**CVS LRA3 Max** Default Value: 1111111b Access: R/W If CVSLRA3Min == CVSLRA3Max , GATR LRA is disabled, GATR cycles are mapped to CVSLRA0 If CVSLRA3Min == CVSLRA3Max , GATR LRA is disabled, CVSLRA2Max will default to CVSLRA3Max to reuse GATR entries
	23	**Reserved** Default Value: 0b Access: RO
	22:16	**CVS LRA3 Min** Default Value: 1110001b Access: R/W
	15	**Reserved** Default Value: 0b Access: RO
	14:8	**CVS LRA2 Max** Default Value: 1110000b Access: R/W Maximum value of programmable LRA2.
	7	**Reserved** Default Value: 0b Access: RO

CVS_TLB_LRA_1 - CVS TLB LRA 1

	6:0	**CVS LRA2 Min** Default Value: 1000000b Access: R/W Minimum value of programmable LRA2.

CVS TLB LRA 2

CVS_TLB_LRA_2 - CVS TLB LRA 2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000031A
Size (in bits):	32
Address:	04A28h

DWord	Bit	Description
0	31:10	**Reserved** Default Value: 00000000000000000000b Access: RO
	9:8	**GATR LRA** Default Value: 11b Access: R/W Which LRA should RS use
	7:6	**RS LRA** Default Value: 00b Access: R/W Which LRA should RS use
	5:4	**CS LRA** Default Value: 01b Access: R/W Which LRA should CS use.
	3:2	**SOL LRA** Default Value: 10b Access: R/W Which LRA should SOL use.
	1:0	**VF LRA** Default Value: 10b Access: R/W Which LRA should VF use.

DATAM

DATAM	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	60030h-60033h
Name:	Transcoder A Data M Value 1
ShortName:	TRANS_DATAM1_A
Power:	PG2
Reset:	soft
Address:	61030h-61033h
Name:	Transcoder B Data M Value 1
ShortName:	TRANS_DATAM1_B
Power:	PG2
Reset:	soft
Address:	62030h-62033h
Name:	Transcoder C Data M Value 1
ShortName:	TRANS_DATAM1_C
Power:	PG2
Reset:	soft
Address:	6F030h-6F033h
Name:	Transcoder EDP Data M Value 1
ShortName:	TRANS_DATAM1_EDP
Power:	PG1
Reset:	soft

Description
There is one instance of this register for each transcoder A/B/C/EDP. This register is double buffered to update on the next MSA after LINKN is written.

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30:25	**TU or VCpayload Size** In DisplayPort SST mode this field is the size of the transfer unit, minus one. Typically it is programmed with a value of 63 for TU size of 64.

DATAM

		In DisplayPort MST mode this field is the Virtual Channel payload size, minus one. **Restriction** Restriction : In DisplayPort MST mode the Virtual Channel payload size must not be programmed greater than 62 (resulting payload size of 63). In DisplayPort MST mode the Virtual Channel payload size must not be changed while the Virtual Channel is enabled, even after a transcoder has been disabled.
	24	**Reserved** Format: MBZ
	23:0	**Data M value** This field is the data M value for internal use.

DATAN

DATAN	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	60034h-60037h
Name:	Transcoder A Data N Value 1
ShortName:	TRANS_DATAN1_A
Power:	PG2
Reset:	soft
Address:	61034h-61037h
Name:	Transcoder B Data N Value 1
ShortName:	TRANS_DATAN1_B
Power:	PG2
Reset:	soft
Address:	62034h-62037h
Name:	Transcoder C Data N Value 1
ShortName:	TRANS_DATAN1_C
Power:	PG2
Reset:	soft
Address:	6F034h-6F037h
Name:	Transcoder EDP Data N Value 1
ShortName:	TRANS_DATAN1_EDP
Power:	PG1
Reset:	soft

Description

There is one instance of this register for each transcoder A/B/C/EDP. This register is double buffered to update on the next MSA after LINKN is written.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:0	**Data N value** This field is the data N value for internal use.

DBUF_CTL

DBUF_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	45008h-4500Bh
Name:	DBUF Control
ShortName:	DBUF_CTL
Power:	PG0
Reset:	soft

DWord	Bit	Description
0	31	**DBUF Power Request** Access: R/W This field requests DBUF power to enable or disable. 0b – Disable 1b – Enable **Programming Notes** DBUF power must be enabled prior to using internal display engine features. Enable power by programming the power request to 1, then wait for the power state to indicate it is enabled.
	30	**DBUF Power State** Access: RO This field indicates the status of DBUF power. 0b – Disabled 1b – Enabled
	29:28	**Reserved** Format: MBZ
	27	**Reserved**
	26	**Reserved** Format: MBZ
	25:24	**Reserved**

DBUF_CTL		
	23:5	**Reserved** Format: MBZ
	4:0	**Reserved**

DBUF_ECC_STAT

DBUF_ECC_STAT	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/WC
Size (in bits):	32
Address:	45010h-45013h
Name:	DBUF ECC Status
ShortName:	DBUF_ECC_STAT
Power:	PG0
Reset:	soft

Each of these fields is a sticky bit that gives the ECC error status for a particular memory bank.
A '1' in a bit indicates that ECC detected an error. A write of '1' to a bit will clear the bit.
Single errors are corrected by ECC. Double errors are not correctable.

DWord	Bit	Description
0	31	**Double Error Bank 15**
	30	**Double Error Bank 14**
	29	**Double Error Bank 13**
	28	**Double Error Bank 12**
	27	**Double Error Bank 11**
	26	**Double Error Bank 10**
	25	**Double Error Bank 9**
	24	**Double Error Bank 8**
	23	**Double Error Bank 7**
	22	**Double Error Bank 6**
	21	**Double Error Bank 5**
	20	**Double Error Bank 4**
	19	**Double Error Bank 3**
	18	**Double Error Bank 2**
	17	**Double Error Bank 1**
	16	**Double Error Bank 0**
	15	**Single Error Bank 15**
	14	**Single Error Bank 14**
	13	**Single Error Bank 13**
	12	**Single Error Bank 12**

DBUF_ECC_STAT		
	11	**Single Error Bank 11**
	10	**Single Error Bank 10**
	9	**Single Error Bank 9**
	8	**Single Error Bank 8**
	7	**Single Error Bank 7**
	6	**Single Error Bank 6**
	5	**Single Error Bank 5**
	4	**Single Error Bank 4**
	3	**Single Error Bank 3**
	2	**Single Error Bank 2**
	1	**Single Error Bank 1**
	0	**Single Error Bank 0**

DC_STATE_EN

DC_STATE_EN

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	45504h-45507h
Name:	Display C State Enable
ShortName:	DC_STATE_EN
Power:	PG0
Reset:	global

DWord	Bit	Description
0	31:10	**Reserved**
	9	**In CSR Flow** Value: 0b — Name: Not In CSR Value: 1b — Name: In CSR **Restriction** Restriction : This field is used for hardware communication. Software must not change this field.
	8	**Block Outbound Traffic** Access is read/write, but hardware can also clear the value based on the PM Request. Value: 0b — Name: Do Not Block Value: 1b — Name: Block **Restriction** Restriction : This field is used for hardware communication. Software must not change this field.
	7:5	**Reserved**
	4	**Mask Poke** This field masks the poke signal that would otherwise be generated by a write to the DC_STATE_SEL register. Value — Name

DC_STATE_EN

		0b: Unmask 1b: Mask **Restriction** Restriction : This field is used for hardware communication. Software must not change this field.
	3	**Reserved**
	2	**Reserved**
	1:0	**Dynamic DC State Enable** This field enables hardware to dynamically enter and exit Display C states. 00b: Disable 01b: Enable up to DC5 10b: Enable up to DC6 **Restriction** Restriction : The Display CSR code must be loaded before this field is enabled.

DDI_AUX_CTL

DDI_AUX_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000023F [KBL]
Access:	R/W
Size (in bits):	32
Address:	64010h-64013h
Name:	DDI A AUX Channel Control
ShortName:	DDI_AUX_CTL_A
Power:	PG1
Reset:	soft
Address:	64110h-64113h
Name:	DDI B AUX Channel Control
ShortName:	DDI_AUX_CTL_B
Power:	PG2
Reset:	soft
Address:	64210h-64213h
Name:	DDI C AUX Channel Control
ShortName:	DDI_AUX_CTL_C
Power:	PG2
Reset:	soft
Address:	64310h-64313h
Name:	DDI D AUX Channel Control
ShortName:	DDI_AUX_CTL_D
Power:	PG2
Reset:	soft

DWord	Bit	Description
0	31	**Send Busy** Access: R/W Set Writing this bit with 1b initiates the transaction, when read this bit will be a 1b until the transmission completes. The transaction is completed when the response is received or when a timeout occurs. This is a sticky bit. Write a 1b to this bit to set it and initiate the transaction. Hardware will clear it when the transaction completes. **Restriction** Restriction : Do not change any fields while Send Busy is asserted.

DDI_AUX_CTL

	Do not write a 1b again until transaction completes. Restriction : Mutex must be acquired through DDI_AUX_MUTEX before initiating an AUX channel transaction.
30	**Done** Access: R/WC A sticky bit that indicates the transaction has completed. Write a 1 to this bit to clear the event

Value	Name
0b	Not done
1b	Done

29	**Interrupt on Done** Access: R/W Enable an interrupt when the transaction completes or times out.

Value	Name
0b	Enable
1b	Disable

28	**Time out error** Access: R/WC A sticky bit that indicates the transaction has timed out. Write a 1 to this bit to clear the event.

Value	Name
0b	Not error
1b	Error

27:26	**Time out timer value** Access: R/W Used to determine how long to wait for receiver response before timing out.

Value	Name
01b	600us
10b	800us
11b	1600us

25	**Receive error** Access: R/WC A sticky bit that indicates that the data received was corrupted, not in multiples of a full byte, or more than 20 bytes. Write a 1 to this bit to clear the event.

Value	Name
0b	Not Error

DDI_AUX_CTL

	1b	Error

24:20 Message Size

Access:	Write/Read Status

The value written to this field indicates the total number bytes to transmit (including the header).
The value read from this field indicates the number of bytes received, including the header, in the last transaction transaction.
Sync/Stop are not part of the message or the message size.
Reads of this field will give the response message size.
The read value will not be valid while Send/Busy bit 31 is asserted.

Restriction
Restriction : Message sizes of 0 or >20 are not allowed. Reads and writes are valid only when the done bit is set and timeout or receive error has not occurred.

19:16 Reserved

15 Reserved

14 Reserved

13 Reserved

12 Reserved

11 Reserved

10 Reserved

9:5 Fast Wake Sync Pulse Count

Default Value:	1 0001b 18 pulses
Access:	R/W

This field determines the total number of SYNC pulses sent during the SYNC phase of a fast wake transaction.
The value programmed is the number of pulses minus 1. When this is field is set to "n" the aux controller will send "n+1" SYNC pulses before transmitting the STOP pattern.

4:0 Sync Pulse Count

Default Value:	1 1111b 32 pulses
Access:	R/W

This field determines the total number of SYNC pulses sent during the SYNC phase of a standard transaction.
The value programmed is the number of pulses minus 1. When this is field is set to "n" the aux controller will send "n+1" SYNC pulses before transmitting the STOP pattern.

Restriction
Restriction : This field must be programmed to at least 25 decimal to send the minimum amount of pulses required for a standard transaction.

DDI_AUX_DATA

DDI_AUX_DATA	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Write/Read Status
Size (in bits):	32
Address:	64014h-64027h
Name:	DDI A AUX Channel Data
ShortName:	DDI_AUX_DATA_A_*
Power:	PG1
Reset:	soft
Address:	64114h-64127h
Name:	DDI B AUX Channel Data
ShortName:	DDI_AUX_DATA_B_*
Power:	PG2
Reset:	soft
Address:	64214h-64227h
Name:	DDI C AUX Channel Data
ShortName:	DDI_AUX_DATA_C_*
Power:	PG2
Reset:	soft
Address:	64314h-64327h
Name:	DDI D AUX Channel Data
ShortName:	DDI_AUX_DATA_D_*
Power:	PG2
Reset:	soft

There are 5 DWords of this register format per instance.

DWord	Bit	Description
0	31:0	**AUX CH DATA** This field contains a DWord of the AUX message. Writes to this register give the data to transmit during the transaction. The MSbyte is transmitted first. Reads to this register will give the response data after transaction complete. The read value will not be valid while the Aux Channel Control Register Send/Busy bit is asserted

DDI_AUX_MUTEX

DDI_AUX_MUTEX	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6402Ch-6402Fh
Name:	DDI A AUX Channel MUTEX
ShortName:	DDI_AUX_MUTEX_A
Power:	PG1
Reset:	soft
Address:	6412Ch-6412Fh
Name:	DDI B AUX Channel MUTEX
ShortName:	DDI_AUX_MUTEX_B
Power:	PG2
Reset:	soft
Address:	6422Ch-6422Fh
Name:	DDI C AUX Channel MUTEX
ShortName:	DDI_AUX_MUTEX_C
Power:	PG2
Reset:	soft
Address:	6432Ch-6432Fh
Name:	DDI D AUX Channel MUTEX
ShortName:	DDI_AUX_MUTEX_D
Power:	PG2
Reset:	soft

Programming Notes
Follow programming sequence from DDI Aux Channel page

DWord	Bit	Description
0	31	**Mutex Enable** Access: R/W This field enables the Mutex. When enabled, mutex allows only one source to use the Aux controller at a time. Value / Name

DDI_AUX_MUTEX

		1b: Enable 0b: Disable
	30	**Mutex Status** Access: Write/Read Status This field indicates the Mutex status. Software must acquire mutex before initiating an Aux transaction. Sticky bit set to 1 after a read to this register when Mutex is enabled. Clear by writing with a 1. **Value** / **Name** 0b: Mutex not yet acquired 1b: Mutex already acquired **Restriction** **After completing an Aux channel transaction with mutex enabled, write this bit with a 1 to clear it so hardware can use Aux for other purposes.**
	29:0	**Reserved**

DDI_BUF_CTL

DDI_BUF_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	64000h-64003h
Name:	DDI A Buffer Control
ShortName:	DDI_BUF_CTL_A
Power:	PG1
Reset:	soft
Address:	64100h-64103h
Name:	DDI B Buffer Control
ShortName:	DDI_BUF_CTL_B
Power:	PG1
Reset:	soft
Address:	64200h-64203h
Name:	DDI C Buffer Control
ShortName:	DDI_BUF_CTL_C
Power:	PG1
Reset:	soft
Address:	64300h-64303h
Name:	DDI D Buffer Control
ShortName:	DDI_BUF_CTL_D
Power:	PG1
Reset:	soft
Address:	64400h-64403h
Name:	DDI E Buffer Control
ShortName:	DDI_BUF_CTL_E
Power:	PG1
Reset:	soft

There is one DDI Buffer Control per each DDI A/B/C/D/E/F.

Do not read or write the register when the associated power well is disabled.

DWord	Bit	Description

DDI_BUF_CTL

0 / 31 — DDI Buffer Enable

This bit enables the DDI buffer.

Value	Name
0b	Disable
1b	Enable

30 — Reserved

Format:	MBZ

29:28 — Reserved

27:24 — DP Vswing Emp Sel

Description
These bits are used to select the voltage swing and emphasis for DisplayPort.
This field is ignored for HDMI and DVI. The values programmed in DDI_BUF_TRANS determine the voltage swing and emphasis for each selection.

Value	Name	Description
0000b-1000b	Select 0 - Select 8	Select from buffer translations 0 through 8. Valid with all DDIs.
1001b	Select 9	Select buffer translation 9. Valid only with DDIA and DDIE.

23:17 — Reserved

Format:	MBZ

16 — Port Reversal

This field enables lane reversal within the port. Lane reversal swaps the data on the lanes as they are output from the port.

Value	Name
0b	Not reversed
1b	Reversed

Programming Notes
DDI B, C, D, and F reversal always swaps the four lanes, so lane 0 is swapped with lane 3, and lane 1 is swapped with lane 2. If DDIA Lane Capability Control selects DDIA x2, then DDI A reversal swaps the two lanes, so lane 0 is swapped with lane 1. If DDIA Lane Capability Control selects DDIA x4, then DDI A reversal swaps the four lanes, so lane 0 is swapped with lane 3, and lane 1 is swapped with lane 2.

Restriction
Restriction : This field must not be changed while the DDI is enabled. DDI E does not support reversal.

DDI_BUF_CTL

15:8 Reserved

Format:	MBZ

7 DDI Idle Status

Access:	RO

This bit indicates when the DDI buffer is idle.

Value	Name
0b	Buffer Not Idle
1b	Buffer Idle

6:5 Reserved

Format:	MBZ

4 DDIA Lane Capability Control

This bit selects how lanes are shared between DDI A and DDI E.
This field is only used in the DDI A instance of this register.
See the DDI A and DDI E lane mapping table in the Introduction section.

Value	Name	Description
0b	DDIA x2	DDI A supports 2 lanes and DDI E supports 2 lanes
1b	DDIA x4	DDI A supports 4 lanes and DDI E is not used

Restriction
Restriction : This field must be programmed at system boot based on board configuration and may not be changed afterwards.

3:1 DP Port Width Selection

Description
This bit selects the number of lanes to be enabled on the DDI link for DisplayPort.

Value	Name	Description
000b	x1	x1 Mode
001b	x2	x2 Mode
011b	x4	x4 Mode Not allowed with DDI E, some restrictions with DDI A
Others	Reserved	Reserved

Programming Notes
DDI E only supports x1 and x2 when DDI_BUF_CTL_A DDIA Lane Capability Control is set to DDIA x2, otherwise DDI E is not supported. DDI A (EDP) supports x1, x2, and x4 when DDI_BUF_CTL_A DDIA Lane Capability Control is set to DDIA x4, otherwise DDI A only supports x1 and x2.

DDI_BUF_CTL

Restriction
Restriction : When in DisplayPort mode the value selected here must match the value selected in TRANS_DDI_FUNC_CTL attached to this DDI.
Restriction : This field must not be changed while the DDI is enabled.

0 Init Display Detected

Access:	RO

Strap indicating whether a display was detected on this port during initialization. It signifies the level of the port detect pin at boot. This bit is only informative. It does not prevent this port from being enabled in hardware. This field only indicates the DDIA detection. Detection for other ports is read from SFUSE_STRAP.

Value	Name	Description
0b	Not Detected	Digital display not detected during initialization
1b	Detected	Digital display detected during initialization

DDI_BUF_TRANS

DDI_BUF_TRANS	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000018, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	64E00h-64E4Fh
Name:	DDI A Buffer Translation
ShortName:	DDI_BUF_TRANS_A_*
Power:	PG1
Reset:	global
Address:	64E60h-64EAFh
Name:	DDI B Buffer Translation
ShortName:	DDI_BUF_TRANS_B_*
Power:	PG1
Reset:	global
Address:	64EC0h-64F0Fh
Name:	DDI C Buffer Translation
ShortName:	DDI_BUF_TRANS_C_*
Power:	PG1
Reset:	global
Address:	64F20h-64F6Fh
Name:	DDI D Buffer Translation
ShortName:	DDI_BUF_TRANS_D_*
Power:	PG1
Reset:	global
Address:	64F80h-64FCFh
Name:	DDI E Buffer Translation
ShortName:	DDI_BUF_TRANS_E_*
Power:	PG1
Reset:	global

Description
These registers define the DDI buffer settings required for different voltage swing and emphasis selections. In HDMI or DVI mode the HDMI/DVI translation registers are automatically selected.
In DisplayPort mode the DDI Buffer Control register programming will select which of these registers is used to

DDI_BUF_TRANS

drive the buffer.
For each DDI A/B/C/D/E there are 10 instances of this 2 DWord register format.
For DDI B/C/D, the first 9 instances (18 Dwords) are entries 0-8 which are used for DisplayPort, and the last instance (2 Dwords) is entry 9 which is used for HDMI and DVI.
For DDI A and DDI E, the 10 instances (20 DWords) are entries 0-9 which are used for DisplayPort.

Programming Notes

The recommended values are listed below this table.

Restriction

Restriction : These registers must be programmed with valid values prior to enabling DDI_BUF_CTL.

DWord	Bit	Description
0	31	**Balance Leg Enable** This field controls the Balance Leg enable for the DDI buffer. Value: 0b — Name: Disable Value: 1b — Name: Enable
	30:18	**Reserved** Format: MBZ
	17:0	**DeEmp Level** Default Value: 00018h This field controls the De-emphasis level for the DDI buffer.
1	31:21	**Reserved** Format: MBZ
	20:16	**VRef Sel** This field controls the voltage reference select for the DDI buffer. Value: 00000b — Name: **[Default]**
	15:11	**Reserved** Format: MBZ
	10:0	**Vswing** Default Value: 00000000000b This field controls the voltage swing for the DDI buffer.

DE_PIPE_INTERRUPT

DE_PIPE_INTERRUPT	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44400h-4440Fh
Name:	Display Engine Pipe A Interrupts
ShortName:	DE_PIPE_INTERRUPT_A
Power:	PG1
Reset:	soft
Address:	44410h-4441Fh
Name:	Display Engine Pipe B Interrupts
ShortName:	DE_PIPE_INTERRUPT_B
Power:	PG2
Reset:	soft
Address:	44420h-4442Fh
Name:	Display Engine Pipe C Interrupts
ShortName:	DE_PIPE_INTERRUPT_C
Power:	PG2
Reset:	soft

Description

This table indicates which events are mapped to each bit of the Display Engine Pipe Interrupt registers.
The IER enabled Display Engine Pipe Interrupt IIR (sticky) bits are ORed together to generate the DE_Pipe Interrupts Pending bit in the Master Interrupt Control register.
There is one full set of Display Engine Pipe interrupts per display pipes A/B/C.
The STEREO3D_EVENT_MASK selects between left eye and right eye reporting of vertical blank, vertical sync, and scanline events in stereo 3D modes.

0x44400 = ISR A, 0x44410 = ISR B, 0x44420 = ISR C
0x44404 = IMR A, 0x44414 = IMR B, 0x44424 = IMR C
0x44408 = IIR A, 0x44418 = IIR B, 0x44428 = IIR C
0x4440C = IER A, 0x4441C = IER B, 0x4442C = IER C

DWord	Bit	Description
0	31	**Underrun** The ISR is an active high pulse when there is an underrun on the transcoder attached to this pipe.
	30	**Unused_Int_30** These interrupts are currently unused.

DE_PIPE_INTERRUPT

	29	**Reserved**
	28	**Reserved**
	27:20	**Unused_Int_27_20** These interrupts are currently unused.
	19	**Reserved**
	18	**Reserved**
	17	**Reserved**
	16	**Reserved**
	15:13	**Unused_Int_15_13** These interrupts are currently unused.
	12	**DPST_Histogram_event** The ISR is an active high pulse on the DPST Histogram event on this pipe.
	11	**Cursor_GTT_Fault_Status** The ISR is an active high pulse when a GTT fault is detected for the cursor on this pipe.
	10	**Plane4_GTT_Fault_Status** **Description** The ISR is an active high pulse when a GTT fault is detected for plane 4 on this pipe. Not all pipes a have plane 4.
	9	**Plane3_GTT_Fault_Status** **Description** The ISR is an active high pulse when a GTT fault is detected for plane 3 on this pipe. Not all pipes a have plane 3.
	8	**Plane2_GTT_Fault_Status** The ISR is an active high pulse when a GTT fault is detected for plane 2 on this pipe.
	7	**Plane1_GTT_Fault_Status** The ISR is an active high pulse when a GTT fault is detected for plane 1 on this pipe.
	6	**Plane4_Flip_Done** **Description** The ISR is an active high pulse when the flip is done for plane 4 on this pipe. Not all pipes have a plane 4.
	5	**Plane3_Flip_Done** **Description** The ISR is an active high pulse when the flip is done for plane 3 on this pipe. Not all pipes have a plane 3.
	4	**Plane2_Flip_Done** The ISR is an active high pulse when the flip is done for plane 2 on this pipe.
	3	**Plane1_Flip_Done** The ISR is an active high pulse when the flip is done for plane 1 on this pipe.

DE_PIPE_INTERRUPT		
	2	**Scan_Line_Event** The ISR is an active high pulse on the scan line event of the transcoder attached to this pipe.
	1	**Vsync** The ISR is an active high level for the duration of the vertical sync of the transcoder attached to this pipe.
	0	**Vblank** The ISR is an active high level for the duration of the vertical blank of the transcoder attached to this pipe.

DE_RR_DEST

DE_RR_DEST	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	44058h-4405Bh
Name:	Render Response Destination
ShortName:	DE_RR_DEST
Power:	PG0
Reset:	soft

This register selects the destination of certain render responses that may go to CS, BCS, or both.
In order for a response to be sent to a particular destination, the event most occur, the event must be unmasked, and that destination must be selected.

DWord	Bit	Description
0	31:6	**Reserved** Format: MBZ
	5:4	**Pipe C Vertical Blank Destination** This field selects the destination for the render response sent on pipe C start of vertical blank. Value / Name: 00b CS; 01b BCS; 10b,11b Both CS and BCS
	3:2	**Pipe B Vertical Blank Destination** This field selects the destination for the render response sent on pipe B start of vertical blank. Value / Name: 00b CS; 01b BCS; 10b,11b Both CS and BCS
	1:0	**Pipe A Vertical Blank Destination** This field selects the destination for the render response sent on pipe A start of vertical blank. Value / Name: 00b CS; 01b BCS; 10b,11b Both CS and BCS

DE_RRMR

DE_RRMR

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x2077EFEF
Access:	R/W
Size (in bits):	32
Address:	44050h-44053h
Name:	Render Response Mask
ShortName:	DE_RRMR
Power:	PG0
Reset:	soft

This register contains a bit mask which selects which events cause and are reported in the render response message.
See the render response message definition table to find the source event for each bit.
The render response message is sent from the display engine to the render command streamer (CS) or blitter command streamer (BCS). The message is used to inform CS and BCS of certain display events.
This register is used to control which render response message bits are masked or unmasked.
Unmasked bits will cause a render response message to be sent and will be reported in that message.
Masked bits will not be reported and will not cause a render response message to be sent.
Vertical blank events occur periodically while the associated display pipe timing generator is running and will be reported in a render response to CS or BCS (depending on DE_RR_DEST destination selection) if un-masked here.
Scanline events occur after they have been initiated through MMIO writes or LRI to the Display Load Scan Lines register.
A flip event will be reported in a render response to CS if un-masked here and the Display Load Scanline source is CS.
A flip event will be reported in a render response to BCS if un-masked here and the Display Load Scanline source is BCS.
Flip done events occur after they have been initiated through MI_DISPLAY_FLIP or MMIO write to plane surface address registers.
A flip event will be reported in a render response to CS if un-masked here and the flip source is CS.
A flip event will be reported in a render response to BCS if un-masked here and the flip source is BCS.

Programming Notes

Programming this register can be done through MMIO or a command streamer LOAD_REGISTER_IMMEDIATE (LRI) command.
When using LRI care must be taken to follow all the programming rules for LRI targetting the display engine.
Unmasked events will wake GT as they occur, so for improved power savings it is recommended to only unmask events that are required.

Restriction

Restriction : Events must be unmasked prior to waiting for them with a MI_WAIT_FOR_EVENT ring command, or

DE_RRMR

in the case of flips or scanlines, prior to starting the flip or loading the scanline.

DWord	Bit	Description
0	31:30	**Reserved** Format: MBZ
	29	**Mask 29** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	28:23	**Reserved**
	22	**Mask 22** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	21	**Mask 21** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	20	**Mask 20** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	19	**Reserved**
	18	**Mask 18** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	17	**Mask 17** Value: 0b – Not Masked Value: 1b – Masked **[Default]**
	16	**Mask 16** Value: 0b – Not Masked Value: 1b – Masked **[Default]**

DE_RRMR

	15	**Mask 15** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	14	**Mask 14** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	13	**Mask 13** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	12	**Reserved**
	11	**Mask 11** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	10	**Mask 10** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	9	**Mask 9** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	8	**Mask 8** Value: 0b — Not Masked Value: 1b — Masked **[Default]**
	7	**Mask 7** Value: 0b — Not Masked Value: 1b — Masked **[Default]**

DE_RRMR

	6	**Mask 6** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**
	5	**Mask 5** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**
	4	**Reserved**
	3	**Mask 3** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**
	2	**Mask 2** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**
	1	**Mask 1** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**
	0	**Mask 0** Value: 0b — Name: Not Masked Value: 1b — Name: Masked **[Default]**

Decouple Register 0 DW0

DECOUPREG0DW0 - Decouple Register 0 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F00h-00F03h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg0DW0Data** Access: R/W Decouple Register 0 DW0 Data.

Decouple Register 0 DW1

DECOUPREG0DW1 - Decouple Register 0 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F04h-00F07h	
DWord	**Bit**	**Description**
0	31	**DecoupReg0DW1GO** Access: R/W Decouple Register 0 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg0DW1PD** Access: R/W Decouple Register 0 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg0DW1OP** Access: R/W Decouple Register 0 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg0DW1BE** Access: R/W Decouple Register 0 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg0DW1Addr** Access: R/W Decouple Register 0 DW1, Decoupled Address.

Decouple Register 1 DW0

DECOUPREG1DW0 - Decouple Register 1 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F08h-00F0Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg1DW0Data** Access: R/W Decouple Register 1 DW0 Data.

Decouple Register 1 DW1

DECOUPREG1DW1 - Decouple Register 1 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F0Ch-00F0Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg1DW1GO** Access: R/W Decouple Register 1 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg1DW1PD** Access: R/W Decouple Register 1 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg1DW1OP** Access: R/W Decouple Register 1 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg1DW1BE** Access: R/W Decouple Register 1 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg1DW1Addr** Access: R/W Decouple Register 1 DW1, Decoupled Address.

Decouple Register 2 DW0

DECOUPREG2DW0 - Decouple Register 2 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F10h-00F13h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg2DW0Data** Access: R/W Decouple Register 2 DW0 Data.

Decouple Register 2 DW1

DECOUPREG2DW1 - Decouple Register 2 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F14h-00F17h	
DWord	**Bit**	**Description**
0	31	**DecoupReg2DW1GO** Access: R/W Decouple Register 2 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg2DW1PD** Access: R/W Decouple Register 2 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg2DW1OP** Access: R/W Decouple Register 2 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg2DW1BE** Access: R/W Decouple Register 2 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg2DW1Addr** Access: R/W Decouple Register 2 DW1, Decoupled Address.

Decouple Register 3 DW0

DECOUPREG3DW0 - Decouple Register 3 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F18h-00F1Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg3DW0Data** Access: R/W Decouple Register 3 DW0 Data.

Decouple Register 3 DW1

DECOUPREG3DW1 - Decouple Register 3 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F1Ch-00F1Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg3DW1GO** Access: R/W Decouple Register 3 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg3DW1PD** Access: R/W Decouple Register 3 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg3DW1OP** Access: R/W Decouple Register 3 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg3DW1BE** Access: R/W Decouple Register 3 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg3DW1Addr** Access: R/W Decouple Register 3 DW1, Decoupled Address.

Decouple Register 4 DW0

DECOUPREG4DW0 - Decouple Register 4 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F20h-00F23h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg4DW0Data** Access: R/W Decouple Register 4 DW0 Data.

Decouple Register 4 DW1

DECOUPREG4DW1 - Decouple Register 4 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F24h-00F27h	
DWord	**Bit**	**Description**
0	31	**DecoupReg4DW1GO** Access: R/W Decouple Register 4 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg4DW1PD** Access: R/W Decouple Register 4 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg4DW1OP** Access: R/W Decouple Register 4 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg4DW1BE** Access: R/W Decouple Register 4 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg4DW1Addr** Access: R/W Decouple Register 4 DW1, Decoupled Address.

Decouple Register 5 DW0

DECOUPREG5DW0 - Decouple Register 5 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F28h-00F2Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg5DW0Data** Access: R/W Decouple Register 5 DW0 Data.

Decouple Register 5 DW1

DECOUPREG5DW1 - Decouple Register 5 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F2Ch-00F2Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg5DW1GO** Access: R/W Decouple Register 5 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg5DW1PD** Access: R/W Decouple Register 5 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg5DW1OP** Access: R/W Decouple Register 5 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg5DW1BE** Access: R/W Decouple Register 5 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg5DW1Addr** Access: R/W Decouple Register 5 DW1, Decoupled Address.

Decouple Register 6 DW0

DECOUPREG6DW0 - Decouple Register 6 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F30h-00F33h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg6DW0Data** Access: R/W Decouple Register 6 DW0 Data.

Decouple Register 6 DW1

DECOUPREG6DW1 - Decouple Register 6 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F34h-00F37h	
DWord	**Bit**	**Description**
0	31	**DecoupReg6DW1GO** Access: R/W Decouple Register 6 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg6DW1PD** Access: R/W Decouple Register 6 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg6DW1OP** Access: R/W Decouple Register 6 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg6DW1BE** Access: R/W Decouple Register 6 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg6DW1Addr** Access: R/W Decouple Register 6 DW1, Decoupled Address.

Decouple Register 7 DW0

DECOUPREG7DW0 - Decouple Register 7 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F38h-00F3Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg7DW0Data** Access: R/W Decouple Register 7 DW0 Data.

Decouple Register 7 DW1

DECOUPREG7DW1 - Decouple Register 7 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F3Ch-00F3Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg7DW1GO** Access: R/W Decouple Register 7 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg7DW1PD** Access: R/W Decouple Register 7 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg7DW1OP** Access: R/W Decouple Register 7 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg7DW1BE** Access: R/W Decouple Register 7 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg7DW1Addr** Access: R/W Decouple Register 7 DW1, Decoupled Address.

Decouple Register 8 DW0

DECOUPREG8DW0 - Decouple Register 8 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F40h-00F43h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg8DW0Data** Access: R/W Decouple Register 8 DW0 Data.

Decouple Register 8 DW1

DECOUPREG8DW1 - Decouple Register 8 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F44h-00F47h	
DWord	**Bit**	**Description**
0	31	**DecoupReg8DW1GO** Access: R/W Decouple Register 8 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg8DW1PD** Access: R/W Decouple Register 8 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg8DW1OP** Access: R/W Decouple Register 8 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg8DW1BE** Access: R/W Decouple Register 8 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg8DW1Addr** Access: R/W Decouple Register 8 DW1, Decoupled Address.

Decouple Register 9 DW0

DECOUPREG9DW0 - Decouple Register 9 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F48h-00F4Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg9DW0Data** Access: R/W Decouple Register 9 DW0 Data.

Decouple Register 9 DW1

DECOUPREG9DW1 - Decouple Register 9 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F4Ch-00F4Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg9DW1GO** Access: R/W Decouple Register 9 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg9DW1PD** Access: R/W Decouple Register 9 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg9DW1OP** Access: R/W Decouple Register 9 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg9DW1BE** Access: R/W Decouple Register 9 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg9DW1Addr** Access: R/W Decouple Register 9 DW1, Decoupled Address.

Decouple Register 10 DW0

DECOUPREG10DW0 - Decouple Register 10 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F50h-00F53h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg10DW0Data** Access: R/W Decouple Register 10 DW0 Data.

Decouple Register 10 DW1

DECOUPREG10DW1 - Decouple Register 10 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F54h-00F57h	
DWord	**Bit**	**Description**
0	31	**DecoupReg10DW1GO** Access: R/W Decouple Register 10 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg10DW1PD** Access: R/W Decouple Register 10 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg10DW1OP** Access: R/W Decouple Register 10 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg10DW1BE** Access: R/W Decouple Register 10 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg10DW1Addr** Access: R/W Decouple Register 10 DW1, Decoupled Address.

Decouple Register 11 DW0

DECOUPREG11DW0 - Decouple Register 11 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F58h-00F5Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg11DW0Data** Access: R/W Decouple Register 11 DW0 Data.

Decouple Register 11 DW1

DECOUPREG11DW1 - Decouple Register 11 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F5Ch-00F5Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg11DW1GO** Access: R/W Decouple Register 11 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg11DW1PD** Access: R/W Decouple Register 11 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg11DW1OP** Access: R/W Decouple Register 11 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg11DW1BE** Access: R/W Decouple Register 11 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg11DW1Addr** Access: R/W Decouple Register 11 DW1, Decoupled Address.

Decouple Register 12 DW0

DECOUPREG12DW0 - Decouple Register 12 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F60h-00F63h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg12DW0Data** Access: R/W Decouple Register 12 DW0 Data.

Decouple Register 12 DW1

DECOUPREG12DW1 - Decouple Register 12 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F64h-00F67h	
DWord	**Bit**	**Description**
0	31	**DecoupReg12DW1GO** Access: R/W Decouple Register 12 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg12DW1PD** Access: R/W Decouple Register 12 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg12DW1OP** Access: R/W Decouple Register 12 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg12DW1BE** Access: R/W Decouple Register 12 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg12DW1Addr** Access: R/W Decouple Register 12 DW1, Decoupled Address.

Decouple Register 13 DW0

DECOUPREG13DW0 - Decouple Register 13 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F68h-00F6Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg13DW0Data** Access: R/W Decouple Register 13 DW0 Data.

Decouple Register 13 DW1

DECOUPREG13DW1 - Decouple Register 13 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F6Ch-00F6Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg13DW1GO** Access: R/W Decouple Register 13 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg13DW1PD** Access: R/W Decouple Register 13 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg13DW1OP** Access: R/W Decouple Register 13 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg13DW1BE** Access: R/W Decouple Register 13 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg13DW1Addr** Access: R/W Decouple Register 13 DW1, Decoupled Address.

Decouple Register 14 DW0

DECOUPREG14DW0 - Decouple Register 14 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F70h-00F73h	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg14DW0Data** Access: R/W Decouple Register 14 DW0 Data.

Decouple Register 14 DW1

DECOUPREG14DW1 - Decouple Register 14 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F74h-00F77h	
DWord	**Bit**	**Description**
0	31	**DecoupReg14DW1GO** Access: R/W Decouple Register 14 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg14DW1PD** Access: R/W Decouple Register 14 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg14DW1OP** Access: R/W Decouple Register 14 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg14DW1BE** Access: R/W Decouple Register 14 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg14DW1Addr** Access: R/W Decouple Register 14 DW1, Decoupled Address.

Decouple Register 15 DW0

DECOUPREG15DW0 - Decouple Register 15 DW0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F78h-00F7Bh	
DWord	**Bit**	**Description**
0	31:0	**DecoupReg15DW0Data** Access: R/W Decouple Register 15 DW0 Data.

Decouple Register 15 DW1

DECOUPREG15DW1 - Decouple Register 15 DW1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00F7Ch-00F7Fh	
DWord	**Bit**	**Description**
0	31	**DecoupReg15DW1GO** Access: R/W Decouple Register 15 DW1 Go_Status Bit, Software writes this bit to 1 when it is ready for the command to be processed, Hardware clears this bit to 0 when the command is complete, Software must not read data or write the next command when this bit is 1.
	30	**Reserved**
	29:28	**DecoupReg15DW1PD** Access: R/W Decouple Register 15 DW1, 00 = GTI/Blitter, 01 = Render, 10= Media , 11= All Domains, Identifies the target power domain that must be awake before proceeding with the cycle.
	27:24	**DecoupReg15DW1OP** Access: R/W Decouple Register 15 DW1, 0000 = Write, 0001 = Read, All others undefined (GT will ignore command and set go/status=0) Decouple Register.
	23:20	**DecoupReg15DW1BE** Access: R/W Decouple Register 15 DW1, Decoupled byte enables. Provided for legacy/workaround purposes only; GT only supports full dword accesses.
	19:18	**Reserved**
	17:0	**DecoupReg15DW1Addr** Access: R/W Decouple Register 15 DW1, Decoupled Address.

DE Misc Interrupt Definition

DE Misc Interrupt Definition	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44460h-4446Fh
Name:	Display Engine Miscellaneous Interrupts
ShortName:	DE_MISC_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the Display Engine Miscellaneous Interrupt registers.
0x44460 = ISR
0x44464 = IMR
0x44468 = IIR
0x4446C = IER

DWord	Bit	Description
0	31	**Poison** The ISR is an active high pulse on receiving the poison response to a memory transaction.
	30	**ECC_Double_Error** The ISR is an active high level while any of the ECC Double Error status bits are set.
	29	**Invalid_GTT_page_table_entry** The ISR is an active high pulse on receiving the iMPH invalid GTT page table entry indication.
	28	**Invalid_page_table_entry_data** The ISR is an active high pulse on receiving the iMPH invalid page table entry data indication.
	27	**GSE** The ISR is an active high pulse on the GSE system level event.
	26	**Camera Interrupt Event** This interrupt is not supported.
	25	**Reserved**
	24	**Reserved**
	23	**WD0_Interrupts_Combined** The ISR is an active high level while any of the WD0_IIR bits are set.
	22	**SVM Device Mode PRQ Event** The ISR is an active high pulse on receiving the iMPH SVM Device Mode PRQ event indication. This event indicates that a GT advanced context encountered a recoverable page fault.
	21	**SVM Device Mode VTD Fault** The ISR is an active high pulse on receiving the iMPH SVM Device Mode VT-d fault indication. This event indicates GT encountered a non-recoverable translation fault.

DE Misc Interrupt Definition		
	20	**SVM Device Mode Wait Descriptor Completion** The ISR is an active high pulse on receiving the iMPH SVM Device Mode Wait Descriptor Completion indication. This event indicates that IMPH completed Invalidation Wait Descriptor.
	19	**SRD_Interrupts_Combined** The ISR is an active high level while any of the SRD_IIR bits are set.
	18	**Reserved** Format: MBZ
	17:16	**Reserved** Format: MBZ
	15	**GTC_Interrupts_Combined** The ISR is an active high level while any of the GTC_IIR bits are set.
	14:9	**Reserved** Format: MBZ
	8	**Reserved** Format: MBZ
	7:4	**Reserved** Format: MBZ
	3:1	**Reserved** Format: MBZ
	0	**Reserved** Format: MBZ

DE Port Interrupt Definition

DE Port Interrupt Definition	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44440h-4444Fh
Name:	Display Engine Port Interrupts
ShortName:	DE_PORT_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the Display Engine Port Interrupt registers.
0x44440 = ISR
0x44444 = IMR
0x44448 = IIR
0x4444C = IER

DWord	Bit	Description
0	31	**MIPI C** The ISR is an active high level indicating an interrupt is set in MIPIC_INTR_STAT_REG or MIPIC_INTR_STAT_REG_1.
	30	**MIPI A** The ISR is an active high level indicating an interrupt is set in MIPIA_INTR_STAT_REG or MIPIA_INTR_STAT_REG_1.
	29	**Reserved**
	28	**Reserved**
	27	**AUX Channel D** The ISR is an active high pulse on the AUX DDI D done event. This event will not occur for SRD AUX done.
	26	**AUX Channel C** The ISR is an active high pulse on the AUX DDI C done event. This event will not occur for SRD AUX done.
	25	**AUX Channel B** The ISR is an active high pulse on the AUX DDI B done event. This event will not occur for SRD AUX done.
	24:23	**Reserved**
	22:12	**Reserved**
	11:10	**Reserved**
	9:8	**Reserved**

DE Port Interrupt Definition		
	7:6	**Reserved**
	5	**DDI C Hotplug** The ISR gives the live state of the DDI HPD pin when the HPD input is enabled. The IIR is set if a short or long pulse is detected when HPD input is enabled. This field is unused in projects that have a PCH.
	4	**DDI B Hotplug** The ISR gives the live state of the DDI HPD pin when the HPD input is enabled. The IIR is set if a short or long pulse is detected when HPD input is enabled. This field is unused in projects that have a PCH.
	3	**DDI A Hotplug** The ISR gives the live state of the DDI HPD pin when the HPD input is enabled. The IIR is set if a short or long pulse is detected when HPD input is enabled. This field is unused in projects that have a PCH.
	2	**Reserved**
	1	**Reserved**
	0	**AUX_Channel_A** The ISR is an active high pulse on the AUX DDI A done event. This event will not occur for SRD AUX done.

Device 2 Control

DEV2CTL_0_2_0_PCI - Device 2 Control		
Register Space:	PCI: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	8	
Address:	00058h	
DWord	**Bit**	**Description**
0	0	**Reserved**

Device Enable

DEVEN_0_0_0_PCI - Device Enable		
Register Space:	PCI: 0/0/0	
Source:	BSpec	
Default Value:	0x000084BF	
Size (in bits):	32	
Address:	00054h	
All the bits in this register are LT Lockable.		

DWord	Bit	Description
0	15	**Device 8 Enable** Default Value: 1b Access: R/W Lock
	14	**Chap Enable** Default Value: 0b Access: R/W
	13	**Device 6 Enable** Default Value: 0b Access: R/W
	12:11	**Reserved** Format: MBZ
	10	**Device 5 Enable** Default Value: 1b Access: R/W Lock
	9:8	**Reserved** Format: MBZ
	7	**Device 4 Enable** Default Value: 1b Access: R/W Lock
	6	**Reserved** Format: MBZ
	5	**Device 3 enable for Display HD Audio** Default Value: 1b Access: R/W Lock

DEVEN_0_0_0_PCI - Device Enable

	4	**Internal Graphics Engine** Default Value: 1b Access: R/W Lock 0: Bus 0 Device 2 is disabled and hidden 1: Bus 0 Device 2 is enabled and visible This bit will be set to 0b and remain 0b if Device 2 capability is disabled.
	3	**PEG10 Enable** Default Value: 1b Access: R/W Lock
	2	**PEG11 Enable** Default Value: 1b Access: R/W Lock
	1	**PEG12 Enable** Default Value: 1b Access: R/W Lock
	0	**Host Bridge** Default Value: 1b Access: RO

Device Identification

DID2_0_2_0_PCI - Device Identification

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00001916
Size (in bits):	16
Address:	00002h

This register combined with the Vendor Identification register uniquely identifies any PCI device.

DWord	Bit	Description
0	15:8	**Device Identification Number MSB** Default Value: 00011001b Access: R/W Firmware Only This is the upper part of a 16 bit value assigned to the Graphics device.
	7:0	**Device Identification Number SKU** Default Value: 00010110b Access: RO Variant Firmware Only These are bits 7:0 of the 16 bit value assigned to the Graphics device.

DFSDONE

DFSDONE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	51080h-51083h
Name:	Display Fuse Done
ShortName:	DFSDONE
Power:	PG0
Reset:	global

This register is not reset by FLR.

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**Download Done** This field indicates when fuse download is complete. 0b – Note Done 1b – Done

DFSM

DFSM

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	51000h-51003h
Name:	Display Fuse
ShortName:	DFSM
Power:	PG0
Reset:	global

This register contains fuse and strap settings for display.
This register is not reset by FLR.

DWord	Bit	Description
0	31	**Internal Graphics Disable** This bit indicates whether internal graphics capability is disabled. Value: 0b – Name: Enable – Description: Internal Graphics Enabled Value: 1b – Name: Disable – Description: Internal Graphics Disabled
	30	**Internal Display Disable** This bit indicates whether the display pipe A (first pipe) capability is disabled. Value: 0b – Name: Enable – Description: Pipe A Capability Enabled Value: 1b – Name: Disable – Description: Pipe A Capability Disabled
	29	**Reserved**
	28	**Display PipeC Disable** This bit indicates whether the display pipe C (third pipe) capability is disabled. Value: 0b – Name: Enable – Description: Pipe C Capability Enabled Value: 1b – Name: Disable – Description: Pipe C Capability Disabled
	27	**Display PM Disable** This bit indicates whether the display power management FBC and DPST capabilities are disabled. Value: 0b – Name: Enable – Description: PM Capability Enabled Value: 1b – Name: Disable – Description: PM Capability Disabled

DFSM

26	**Display eDP Disable** This bit indicates whether the display embedded DisplayPort eDP DDIA capability is disabled.

Value	Name	Description
0b	Enable	eDP Capability Enabled
1b	Disable	eDP Capability Disabled

25	**Reserved**
24:23	**Display CDCLK Limit** This field indicates the maximum allowed CD clock frequency.

Value	Name	Description
00b	675 MHz	Maximum frequency is 675 MHz. All frequencies are allowed.
01b	540 MHz	Maximum frequency is 540 MHz.
10b	450 MHz	Maximum frequency is 450 MHz.
11b	337.5 MHz	Maximum frequency is 337.5 MHz.

Restriction
Restriction : Display software should not select any frequency higher than the maximum that is allowed. If software incorrectly selects a higher frequency, display hardware will internally override the selection to the lowest frequency.

22	**Display Spare**
21	**Spare 21** This bit indicates whether the display pipe B (second pipe) capability is disabled.

Value	Name
0b	Pipe B Capability Enabled
1b	Pipe B Capability Disabled

20	**Display WD Disable** This bit indicates whether the display WD capability is disabled.

Value	Name	Description
0b	Enable	WD Capability Enabled
1b	Disable	WD Capability Disabled

19	**Spare 19**
18	**Spare 18**
17	**Spare 17**
16	**Spare 16**
15	**Spare 15**
14	**Spare 14**
13	**Spare 13**
12	**Spare 12**

DFSM

	11	**Spare 11**
	10	**Spare 10**
	9	**Spare 9**
	8	**Spare 8**
	7	**Spare 7**
	6	**Display RSB Enable** This bit indicates whether the remote screen blanking feature is enabled in the display engine.

Value	Name	Description
0b	Disable	RSB Capability Disabled
1b	Enable	RSB Capability Enabled

	5	**Spare 5**
	4	**Spare 4**
	3	**Spare 3**
	2	**Spare 2**
	1	**Reserved**
	0	**Display Audio Codec Disable** This bit indicates whether the display audio codec capability is disabled.

Value	Name	Description
0b	Enable	Audio Codec Capability Enabled
1b	Disable	Audio Codec Capability Disabled

DISPIO_CR_TX_BMU_CR0

DISPIO_CR_TX_BMU_CR0	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C00Ch-6C00Fh
Name:	DISPIO_CR_TX_BMU_CR0
ShortName:	DISPIO_CR_TX_BMU_CR0
Power:	PG0
Reset:	global

The tx_blnclegsctl values are only used when tx_blnclegdisbl==0x00 and DDI_BUF_TRANS dword 0 bit 31==0x1 for the associated DDI.

DWord	Bit	Description
0	31:29	**digital_analog** Display software must not change this field.
	28	**tx_glb_vs_loc_vref_sel** Display software must not change this field.
	27:23	**tx_blnclegdisbl** Disable balance leg
	22:20	**tx_blnclegsctl_4** Balance leg select DDI4 (DDIE or DDIA with x4 capability)
	19:17	**tx_blnclegsctl_3** Balance leg select DDI3 (DDID)
	16:14	**tx_blnclegsctl_2** Balance leg select DDI2 (DDIC)
	13:11	**tx_blnclegsctl_1** Balance leg select for DDI1 (DDIB)
	10:8	**tx_blnclegsctl_0** Balance leg select for DDI0 (DDIA)
	7:0	**tx_h_mode** Display software must not change this field.

Display CSR Program

Display CSR Program	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Display CSR Program

	0x00000000, 0x00000000, 0x00000000,

Display Message Forward Status Register

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	022E8h-022EBh
Name:	Display Message Forward Status Register
ShortName:	DISPLAY_MESSAGE_FORWARD_STATUS_RCSUNIT

Address:	122E8h-122EBh
Name:	Display Message Forward Status Register
ShortName:	DISPLAY_MESSAGE_FORWARD_STATUS_VCSUNIT0

Address:	1A2E8h-1A2EBh
Name:	Display Message Forward Status Register
ShortName:	DISPLAY_MESSAGE_FORWARD_STATUS_VECSUNIT

Address:	1C2E8h-1C2EBh
Name:	Display Message Forward Status Register
ShortName:	DISPLAY_MESSAGE_FORWARD_STATUS_VCSUNIT1

Address:	222E8h-222EBh
Name:	Display Message Forward Status Register
ShortName:	DISPLAY_MESSAGE_FORWARD_STATUS_BCSUNIT

This register stores the internal HW status flags related to display message forward logic. This register should not be accessed by SW. This register is part of power context image.
Note: Even though this register exists in VideoCS and VideoEnhancementCS, individual bit driven functionality is not supported.

DWord	Bit	Description
0	31:30	**Reserved** Source: RenderCS, BlitterCS Format: MBZ
	29:28	**Reserved**
	27:26	**Reserved**
	25:24	**Reserved**
	23:22	**Reserved**

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register		
	21:20	**Reserved**
	19:18	**Reserved**
	17:16	**Reserved**
	15:14	**Reserved**
	13:12	**Reserved**
	11:10	**Reserved**
	9:8	**Reserved**
	7:6	**Reserved**
	5:4	**Reserved**
	3:2	**Reserved**
	31:0	**Reserved** Source: VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	1:0	**Reserved**

DOUBLE_BUFFER_CTL

DOUBLE_BUFFER_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	44500h-44503h
Name:	Double Buffer Control
ShortName:	DOUBLE_BUFFER_CTL
Power:	PG0
Reset:	soft

This register together with the Allow Double Buffer Disable fields in the plane control registers allows for the double buffer update of registers in multiple resources to be synchronized together for an atomic update.

Programming Notes

Sequence for synchronizing the double buffer updates of multiple resources:
1. Set the Allow Double Buffer Update Disable field for each resource to be synchronized together and write the appropriate register to arm and trigger the update. Set the Global Double Buffer Update Disable field. The order in which these fields are set does not matter.
2. Program the registers that need to be synchronized together.
3. Clear the Global Double Buffer Update Disable field. Any pending updates will take place at the next periodic udate event.
4. If a resource no longer needs to be synchronized, clear the Allow Double Buffer Update Disable field for that resource and write the appropriate register to arm and trigger the update. If the resource will continue to be synchronized, the field can remain set and does not need to be set again when returning to step 1 of this sequence.

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**Global Double Buffer Update Disable** This field controls whether the double buffer update is disabled for the resources which have allowed it to be disabled. This only disables the double buffer update for periodic events, like the start of vertical blank. It does not change the behavior for constant events, like pipe not enabled. This applies to MMIO register updates as well as command streamer initiated flips. When the double buffer update is disabled, the values written into the double buffered registers will not take effect at the periodic update event. After the double buffer update is no longer disabled, any pending updates will take place at the next periodic update event. Asynchronous flips initiated by MMIO or command streamers are not effected by disabling double

DOUBLE_BUFFER_CTL

buffering.
Synchronous flips (regular and stereo 3D) initiated by MMIO or command streamers will not complete or give the flip done indication while double buffering is disabled for a plane. They will complete and give the flip done at the next start of vertical blank (selectable right or left eye vertical blank when using stereo 3D) after the double buffering is re-enabled.

Value	Name
0b	Not Disabled
1b	Disabled

DP_TP_CTL

DP_TP_CTL		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	64040h-64043h	
Name:	DDI A DisplayPort Transport Control	
ShortName:	DP_TP_CTL_A	
Power:	PG1	
Reset:	soft	
Address:	64140h-64143h	
Name:	DDI B DisplayPort Transport Control	
ShortName:	DP_TP_CTL_B	
Power:	PG2	
Reset:	soft	
Address:	64240h-64243h	
Name:	DDI C DisplayPort Transport Control	
ShortName:	DP_TP_CTL_C	
Power:	PG2	
Reset:	soft	
Address:	64340h-64343h	
Name:	DDI D DisplayPort Transport Control	
ShortName:	DP_TP_CTL_D	
Power:	PG2	
Reset:	soft	
Address:	64440h-64443h	
Name:	DDI E DisplayPort Transport Control	
ShortName:	DP_TP_CTL_E	
Power:	PG2	
Reset:	soft	

DWord	Bit	Description
0	31	**Transport Enable** This bit enables the DisplayPort transport function.

DP_TP_CTL

Value	Name
0b	Disable
1b	Enable

30 Reserved

Format:	MBZ

29:28 Reserved

27 Transport Mode Select

Description
This bit selects between DisplayPort SST and MST modes of operation. This bit is ignored by DDI A (EDP) and DDI E since they do not support multistreaming.

Value	Name	Description
0b	SST mode	DisplayPort SST mode
1b	MST mode	DisplayPort MST mode

Restriction
Restriction : The DisplayPort mode (SST or MST) selected here must match the mode selected in the Transcoder DDI Function Control registers for the transcoders attached to this transport. This field must not be changed while the DDI function is enabled.

26 Reserved

Format:	MBZ

25 Force ACT

Description
This bit forces DisplayPort MST ACT to be sent one time at the next link frame boundary. After ACT is sent, as indicated in the ACT sent status bit, this bit can be cleared and set again to send ACT again. This bit is ignored by DDI A (EDP) and DDI E since they do not support multistreaming.

Value	Name	Description
0b	Do not force	Do not force ACT to be sent
1b	Force	Force ACT to be sent one time

24:21 Reserved

Format:	MBZ

20:19 Reserved

Format:	MBZ

DP_TP_CTL

18 Enhanced Framing Enable

Description
This bit selects enhanced framing for DisplayPort SST.
Hardware internally enables enhanced framing for DisplayPort MST.

Value	Name
0b	Disabled
1b	Enabled

Restriction
Restriction : In DisplayPort MST mode this bit must be set to Disabled. This field must not be changed while the DDI function is enabled.

17:16 Reserved

Format:	MBZ

15 Reserved

Format:	MBZ

14:11 Reserved

Format:	MBZ

10:8 DP Link Training Enable

These bits are used for DisplayPort link initialization as defined in the DisplayPort specification. DP_TP_STATUS has an indication that the required number of idle patterns has been sent.

Value	Name	Description
000b	Pattern 1	Training Pattern 1 enabled
001b	Pattern 2	Training Pattern 2 enabled
010b	Idle	Idle Pattern enabled
011b	Normal	Link not in training: Send normal pixels
100b	Pattern 3	Training Pattern 3 enabled
Others	Reserved	Reserved

Restriction
Restriction : When enabling the port, it must be turned on with pattern 1 enabled. When retraining a link, the port must be disabled, then re-enabled with pattern 1 enabled.

7 Reserved

DP_TP_CTL

	6	**Alternate SR Enable** This bit enables the DisplayPort Alternate Scrambler Reset, intended for use only with embedded DisplayPort receivers.

Value	Name
0b	Disable
1b	Enable

Restriction
Restriction : This field must not be changed while the DDI function is enabled.

	5:0	**Reserved**

Format:	MBZ

DP_TP_STATUS

DP_TP_STATUS	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	64144h-64147h
Name:	DDI B DisplayPort Transport Status
ShortName:	DP_TP_STATUS_B
Power:	PG2
Reset:	soft
Address:	64244h-64247h
Name:	DDI C DisplayPort Transport Status
ShortName:	DP_TP_STATUS_C
Power:	PG2
Reset:	soft
Address:	64344h-64347h
Name:	DDI D DisplayPort Transport Status
ShortName:	DP_TP_STATUS_D
Power:	PG2
Reset:	soft
Address:	64444h-64447h
Name:	DDI E DisplayPort Transport Status
ShortName:	DP_TP_STATUS_E
Power:	PG2
Reset:	soft

There is one DisplayPort Transport Status register per each DDI B/C/D/E/F. DDI A does not have a status register.

DWord	Bit	Description
0	31:29	**Reserved** Format: MBZ
	28	**Reserved**
	27	**Idle Link Frame Status** Access: R/WC This bit indicates if a link frame boundary has been sent in idle pattern.

DP_TP_STATUS

This is a sticky bit, cleared by writing 1b to it.

Value	Name
0b	Idle link frame not sent
1b	Idle link frame sent

26 **Active Link Frame Status**

Access:	R/WC

This bit indicates if a link frame boundary has been sent in active (at least one VC enabled).
This is a sticky bit, cleared by writing 1b to it.

Value	Name
0b	Active link frame not sent
1b	Active link frame sent

25 **Min Idles Sent**

Access:	RO

This bit indicates that the minimum required number of idle patterns has been sent when DP_TP_CTL is set to send idle patterns.
This bit will clear itself when DP_TP_CTL is not longer set to send idle patterns.

Value	Name
0b	Min idles not sent
1b	Min idles sent

24 **ACT Sent Status**

Access:	R/WC

This bit indicates if DisplayPort MST ACT has been sent.
This is a sticky bit, cleared by writing 1b to it.

Value	Name
0b	ACT not sent
1b	ACT sent

23 **Mode Status**

Access:	RO

This bit indicates what mode the transport is currently in.

Value	Name	Description
0b	SST	Single-stream mode
1b	MST	Multi-stream mode

22:18 **Reserved**

Format:	MBZ

17:16 **Streams Enabled**

Access:	RO

This field indicates the number of streams (transcoders) enabled on this port during multistream

DP_TP_STATUS

operation.
This field should be ignored in single stream mode.

Value	Name	Description
00b	Zero	Zero streams enabled
01b	One	One stream enabled
10b	Two	Two streams enabled
11b	Three	Three streams enabled

15:13 **Reserved**

Format:	MBZ

12 **Reserved**

Format:	MBZ

11:10 **Reserved**

Format:	MBZ

9:8 **Payload Mapping VC2**

Access:	RO

This field indicates which transcoder is mapped to Virtual Channel 2 during multistream operation.
This field should be ignored if the number of streams enabled is less than three.
This field should be ignored in single stream mode.

Value	Name	Description
00b	A	Transcoder A mapped to this VC
01b	B	Transcoder B mapped to this VC
10b	C	Transcoder C mapped to this VC
11b	Reserved	Reserved

7:6 **Reserved**

Format:	MBZ

5:4 **Payload Mapping VC1**

Access:	RO

This field indicates which transcoder is mapped to Virtual Channel 1 during multistream operation.
This field should be ignored if the number of streams enabled is less than two.
This field should be ignored in single stream mode.

Value	Name	Description
00b	A	Transcoder A mapped to this VC
01b	B	Transcoder B mapped to this VC
10b	C	Transcoder C mapped to this VC
11b	Reserved	Reserved

DP_TP_STATUS

	3:2	**Reserved** Format: MBZ
	1:0	**Payload Mapping VC0** Access: RO This field indicates which transcoder is mapped to Virtual Channel 0 during multistream operation. This field should be ignored if the number of streams enabled is less than one. This field should be ignored in single stream mode.

Value	Name	Description
00b	A	Transcoder A mapped to this VC
01b	B	Transcoder B mapped to this VC
10b	C	Transcoder C mapped to this VC
11b	Reserved	Reserved

DPLL_CFGCR1

DPLL_CFGCR1	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C040h-6C043h
Name:	DPLL1_CFGCR1
ShortName:	DPLL1_CFGCR1
Power:	PG0
Reset:	global
Address:	6C048h-6C04Bh
Name:	DPLL2_CFGCR1
ShortName:	DPLL2_CFGCR1
Power:	PG0
Reset:	global
Address:	6C050h-6C053h
Name:	DPLL3_CFGCR1
ShortName:	DPLL3_CFGCR1
Power:	PG0
Reset:	global

This register, together with DPLL_CFGCR2, is used to configure the frequency for DPLL1, DPLL2, and DPLL3, when DPLL_CTRL1 override is enabled and set to HDMI mode.
This register is not reset by the device 2 FLR.

DWord	Bit	Description
0	31	**Frequency Enable** Programmable HDMI/DVI frequency enable Value: 0b = Disable; 1b = Enable
	30:24	**Reserved** Format: MBZ
	23:9	**DCO Fraction** (DCO Frequency/24 - INT(DCO Frequency/24)) * 2^15
	8:0	**DCO Integer** INT (DCO Frequency/24)

DPLL_CFGCR2

DPLL_CFGCR2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C044h-6C047h
Name:	DPLL1_CFGCR2
ShortName:	DPLL1_CFGCR2
Power:	PG0
Reset:	global
Address:	6C04Ch-6C04Fh
Name:	DPLL2_CFGCR2
ShortName:	DPLL2_CFGCR2
Power:	PG0
Reset:	global
Address:	6C054h-6C057h
Name:	DPLL3_CFGCR2
ShortName:	DPLL3_CFGCR2
Power:	PG0
Reset:	global

This register, together with DPLL_CFGCR1, is used to configure the frequency for DPLL1, DPLL2, and DPLL3, when DPLL_CTRL1 override is enabled and set to HDMI mode.
This register is not reset by the device 2 FLR.

Programming Notes

P0 is P, P1 is Q, P2 is K.
The post divider is P*Q*K (P0*P1*P2).

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:8	**Qdiv Ratio** This field specifies the Q (P1) divider ratio. This field is only used when Qdiv Mode is set to Enable to get a divider value other than 1.
	7	**Qdiv Mode** This field enables the Q (P1) divider when the ratio is not 1. Value / Name / Description

DPLL_CFGCR2

0b	Disable	Q divider = 1
1b	Enable	Q divider = Qdiv Ratio

Restriction
Restriction : If K (P2) is not 2, Q (P1) MUST be 1 to ensure 50% duty cycle.

6:5 **Kdiv**

This field specifies the K (P2) divider ratio.

Value	Name
00b	5
01b	2
10b	3
11b	1

4:2 **Pdiv**

This field specifies the P (P0) divider ratio.

Value	Name	Programming Notes
000b	1	Restriction : P (P0) can only be 1 if Q (P1) is also 1.
001b	2	
010b	3	
100b	7	

1:0 **Central Frequency**

This field specifies the center frequency.

Value	Name
00b	9600 MHz
01b	9000 MHz
10b	Reserved
11b	8400 MHz

DPLL_CTRL1

DPLL_CTRL1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C058h-6C05Bh
Name:	DPLL_CTRL1
ShortName:	DPLL_CTRL1
Power:	PG0
Reset:	global

This register controls the DPLL mode, rate, and SSC.
This register is not reset by the device 2 FLR.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27	**Reserved**
	26	**Reserved**
	25	**Reserved**
	24	**Reserved**
	23	**DPLL3 HDMI Mode** Select between DP and HDMI mode Value / Name / Description: 0b / DP mode / Frequency and SSC programmed in this register. 1b / HDMI mode / Frequency is programmed in DPLL*_CFGCR* registers.
	22	**DPLL3 SSC** SSC enable Value / Name: 0b / Disable 1b / Enable
	21:19	**DPLL3 Link Rate** Link rate for DP mode Value / Name / Description: 000b / 2700 / 2700 MHz (DP 5.4 GHz) - VCO 8100 001b / 1350 / 1350 MHz (DP 2.7 GHz) - VCO 8100

DPLL_CTRL1

010b	810	810 MHz (DP 1.62 GHz) - VCO 8100
011b	1620	1620 MHz (DP 3.24 GHz) - VCO 8100
100b	1080	1080 MHz (DP 2.16 GHz) - VCO 8640
101b	2160	2160 MHz (DP 4.32 GHz) - VCO 8640
110b	Reserved	Reserved
111b	Reserved	Reserved

18 — DPLL3 Override

Programming enable

Value	Name
0b	Disable
1b	Enable

17 — DPLL2 HDMI Mode

Select between DP and HDMI mode

Value	Name	Description
0b	DP mode	Frequency and SSC programmed in this register.
1b	HDMI mode	Frequency is programmed in DPLL*_CFGCR* registers.

16 — DPLL2 SSC

SSC enable

Value	Name
0b	Disable
1b	Enable

15:13 — DPLL2 Link Rate

Link rate for DP mode

Value	Name	Description
000b	2700	2700 MHz (DP 5.4 GHz) - VCO 8100
001b	1350	1350 MHz (DP 2.7 GHz) - VCO 8100
010b	810	810 MHz (DP 1.62 GHz) - VCO 8100
011b	1620	1620 MHz (DP 3.24 GHz) - VCO 8100
100b	1080	1080 MHz (DP 2.16 GHz) - VCO 8640
101b	2160	2160 MHz (DP 4.32 GHz) - VCO 8640
110b	Reserved	Reserved
111b	Reserved	Reserved

12 — DPLL2 Override

Programming enable

Value	Name
0b	Disable

DPLL_CTRL1

	1b	Enable

11 DPLL1 HDMI Mode

Select between DP and HDMI mode

Value	Name	Description
0b	DP mode	Frequency and SSC programmed in this register.
1b	HDMI mode	Frequency is programmed in DPLL*_CFGCR* registers.

10 DPLL1 SSC

SSC enable

Value	Name
0b	Disable
1b	Enable

9:7 DPLL1 Link Rate

Link rate for DP mode

Value	Name	Description
000b	2700	2700 MHz (DP 5.4 GHz) - VCO 8100
001b	1350	1350 MHz (DP 2.7 GHz) - VCO 8100
010b	810	810 MHz (DP 1.62 GHz) - VCO 8100
011b	1620	1620 MHz (DP 3.24 GHz) - VCO 8100
100b	1080	1080 MHz (DP 2.16 GHz) - VCO 8640
101b	2160	2160 MHz (DP 4.32 GHz) - VCO 8640
110b	Reserved	Reserved
111b	Reserved	Reserved

6 DPLL1 Override

Programming enable

Value	Name
0b	Disable
1b	Enable

5 Reserved

4 Reserved

3:1 DPLL0 Link Rate

Link rate for DP mode

Value	Name	Description
000b	2700	2700 MHz (DP 5.4 GHz) - VCO 8100
001b	1350	1350 MHz (DP 2.7 GHz) - VCO 8100
010b	810	810 MHz (DP 1.62 GHz) - VCO 8100
011b	1620	1620 MHz (DP 3.24 GHz) - VCO 8100

DPLL_CTRL1

		100b	1080	1080 MHz (DP 2.16 GHz) - VCO 8640
		101b	2160	2160 MHz (DP 4.32 GHz) - VCO 8640
		110b	Reserved	Reserved
		111b	Reserved	Reserved
	0	**DPLL0 Override** Programming enable		

Value	Name
0b	Disable
1b	Enable

DPLL_CTRL2

DPLL_CTRL2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C05Ch-6C05Fh
Name:	DPLL_CTRL2
ShortName:	DPLL_CTRL2
Power:	PG0
Reset:	global

This register controls the mapping of DPLL to port.
This register is not reset by the device 2 FLR.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23	**Reserved**
	22	**Reserved**
	21	**Reserved**
	20	**Reserved**
	19	**DDIE Clock Off** DDIE (DDI4, EDP2) gate the clock going to the port 0b: On 1b: Off
	18	**DDID Clock Off** DDID (DDI3) gate the clock going to the port 0b: On 1b: Off
	17	**DDIC Clock Off** DDIC (DDI2) gate the clock going to the port 0b: On 1b: Off

DPLL_CTRL2

	16	**DDIB Clock Off** DDIB (DDI1) gate the clock going to the port

Value	Name
0b	On
1b	Off

	15	**DDIA Clock Off** DDIA (DDI0, EDP) gate the clock going to the port

Value	Name
0b	On
1b	Off

	14:13	**DDIE Clock Select** DDIE (DDI4, EDP2) port mux select

Value	Name
00b	DPLL0
01b	DPLL1
10b	DPLL2
11b	DPLL3

	12	**DDIE Select Override** DDIE (DDI4, EDP2) programming enable

Value	Name
0b	Disable
1b	Enable

	11:10	**DDID Clock Select** DDID (DDI3) port mux select

Value	Name
00b	DPLL0
01b	DPLL1
10b	DPLL2
11b	DPLL3

	9	**DDID Select Override** DDID (DDI3) programming enable

Value	Name
0b	Disable
1b	Enable

DPLL_CTRL2

	8:7	**DDIC Clock Select** DDIC (DDI2) port mux select

Value	Name
00b	DPLL0
01b	DPLL1
10b	DPLL2
11b	DPLL3

	6	**DDIC Select Override** DDIC (DDI2) programming enable

Value	Name
0b	Disable
1b	Enable

	5:4	**DDIB Clock Select** DDIB (DDI1) port mux select

Value	Name
00b	DPLL0
01b	DPLL1
10b	DPLL2
11b	DPLL3

	3	**DDIB Select Override** DDIB (DDI1) programming enable

Value	Name
0b	Disable
1b	Enable

	2:1	**DDIA Clock Select** DDIA (DDI0, EDP) port mux select

Value	Name
00b	DPLL0
01b	DPLL1
10b	DPLL2
11b	DPLL3

	0	**DDIA Select Override** DDIA (DDI0, EDP) programming enable

Value	Name
0b	Disable
1b	Enable

DPLL_STATUS

DPLL_STATUS	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	6C060h-6C063h
Name:	DPLL_STATUS
ShortName:	DPLL_STATUS
Power:	PG0
Reset:	global

DWord	Bit	Description
0	31:29	**Reserved** Format: MBZ
	28	**DPLL3 SEM Done** Access: RO Value: 0b – Not Done Value: 1b – Done
	27:25	**Reserved** Format: MBZ
	24	**DPLL3 Lock** Access: RO Value: 0b – Not Locked Value: 1b – Locked
	23:21	**Reserved** Format: MBZ
	20	**DPLL2 SEM Done** Access: RO Value: 0b – Not Done

DPLL_STATUS

		1b: Done
	19:17	**Reserved** Format: MBZ
	16	**DPLL2 Lock** Access: RO 0b: Not Locked 1b: Locked
	15:13	**Reserved** Format: MBZ
	12	**DPLL1 SEM Done** Access: RO 0b: Not Done 1b: Done
	11:9	**Reserved** Format: MBZ
	8	**DPLL1 Lock** Access: RO 0b: Not Locked 1b: Locked
	7:5	**Reserved** Format: MBZ
	4	**DPLL0 SEM Done** Access: RO 0b: Not Done 1b: Done
	3:1	**Reserved** Format: MBZ

DPLL_STATUS

	0	**DPLL0 Lock** Access: RO Value / Name: 0b – Not Locked 1b – Locked

DPST_BIN

DPST_BIN	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Access to this address are steered to the correct register by programming the Bin Register Function Select and the Bin Register Index.
Updates take place at the start of vertical blank.

DWord	Bit	Description
0	31	**Busy Bit** If (DPST_CTL:Bin Register Function Select = Threshold Count) {This is a read only bit. If set, the engine is busy and the rest of the register is undefined. If clear, the register contains valid data.} Else (Image Enhancement) {This bit is reserved.}
	30:24	**Reserved**
	23:0	**Data** If (DPST_CTL : Bin Register Function Select = Threshold Count) {Bits 23:0 are read only bits. They indicate the total number of pixels in this bin. The bin value is updated when guardband interrupt delay is met, and is not valid until after a histogram event has occurred. The bin value will stop incrementing once the maximum has been reached.} Else (Image Enhancement) {Bits 23:10 are reserved and should be written as zeroes. Bits 9:0 are R/W double-buffered and program the correction value for this bin. Writes to this register are double buffered on the next vblank. The value written here is the 10bit corrected channel value for the lowest point of the bin.}

DPST_CTL

DPST_CTL

Register Space: MMIO: 0/2/0
Source: BSpec
Default Value: 0x00000000
Access: R/W
Size (in bits): 32

DWord	Bit	Description
0	31	**IE Histogram Enable** This bit enables the Image Enhancement histogram logic to collect data. The collected data will be valid after a histogram event has occurred. **Value** / **Name** 0b – Disable 1b – Enable **Programming Notes** If histogram is enabled while no planes are enabled on the pipe, it may get an incorrect pixel count for a frame.
	30:28	**Reserved**
	27	**IE Modification Table Enable** This bit enables the Image Enhancement modification table. When enabled, modifications begin after the next vertical blank. **Value** / **Name** 0b – Disable 1b – Enable
	26:25	**Reserved**
	24	**Histogram Mode Select** **Value** / **Name** / **Description** 0b – YUV – YUV Luma Mode 1b – HSV – HSV Intensity Mode
	23:16	**Reserved**
	15	**IE Table Value Format** This field indicates what format is used for the image enhancement table values. **Value** / **Name** / **Description** 0b – 1.9 – 1 integer and 9 fractional bits 1b – 2.8 – 2 integer and 8 fractional bits

DPST_CTL

	14:13	**Enhancement mode** Value / Name / Description 00b / Direct / Direct look up mode 01b / Additive / Additive mode 10b / Multiplicative / Multiplicative mode 11b / Reserved / Reserved
	12	**Reserved**
	11	**Bin Register Function Select** This field indicates what data is being written to or read from the bin data register. Value / Name / Description 0b / TC / Threshold Count. A read from the bin data register returns that bin's threshold value from the most recent vblank load event (guardband threshold trip). Valid range for the Bin Index is 0 to 31. 1b / IE / Image Enhancement Value. Valid range for the Bin Index is 0 to 32
	10:7	**Reserved**
	6:0	**Bin Register Index** This field indicates the bin number whose data can be accessed through the bin data register. This value is automatically incremented by a read or a write to the bin data register if the busy bit is not set.

DPST_GUARD

DPST_GUARD

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank

Updates take place at the start of vertical blank.

DWord	Bit	Description
0	31	**Histogram Interrupt enable** Value / Name / Description: 0b / Disable / Disabled 1b / Enable / This generates a histogram interrupt once a Histogram event occurs.
	30	**Histogram Event status** Access: R/WC When a Histogram event has occurred, this will get set by the hardware. For any more Histogram events to occur, clear this bit by writing a '1'. Value / Name / Description: 0b / Not Occurred / Histogram event has not occurred 1b / Occured / Histogram event has occurred
	29:22	**Guardband Interrupt Delay** An interrupt is always generated after this many consecutive frames of the guardband threshold being surpassed. This value is double buffered on start of vblank. Restriction Restriction : A value of 0 is invalid.
	21:0	**Threshold Guardband** This value is used to determine the guardband for the threshold interrupt generation. This single value is used for all the segments. This value is double buffered on start of vblank. This value is shifted left 2 bits (multiplied by 4) for use with the 24 bit bin values.

Driver Media Force Wake Ack

DRIVER_MEDIA_FWAKE_ACK - Driver Media Force Wake Ack	
Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00D88h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
To set bit0, for example, the data would be 0x0001_0001.
To clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:0	**GPM Driver Media ForceWake Ack** Access: R/W 1'b0 : GT Media Can be powered down (default) 1'b1 : GT Media cannot be powered down

Driver Render Force Wake Ack

DRIVER_RENDER_FWAKE_ACK - Driver Render Force Wake Ack

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00D84h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
To set bit0, for example, the data would be 0x0001_0001.
To clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:0	**GPM Driver ForceWake Ack** Access: R/W 1'b0 : GT Render Can be powered down (default) 1'b1 : GT Render cannot be powered down

DS Invocation Counter

DS_INVOCATION_COUNT - DS Invocation Counter	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02308h

This register stores the number of domain points shaded by the DS threads. Domain points which hit in the DS cache will not cause this register to increment. Note that the spawning of a DS thread which shades two domain points will cause this counter to increment by two.This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**DS Invocation Count UDW** Number of domain points shaded by the DS threads. Updated only when DS Function Enable and Statistics Enable are set in 3DSTATE_DS
	31:0	**DS Invocation Count LDW** Number of domain points shaded by the DS threads. Updated only when DS Function Enable and Statistics Enable are set in 3DSTATE_DS

DSSM

DSSM	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	51004h-51007h
Name:	Display Strap State
ShortName:	DSSM
Power:	PG0
Reset:	global

This register contains fuse and strap settings for display.
This register is not reset by FLR.

DWord	Bit	Description
0	31	**Spare 31**
	30	**Spare 30**
	29	**Spare 29**
	28	**Spare 28**
	27	**Spare 27**
	26	**Spare 26**
	25	**Spare 25**
	24	**Spare 24**
	23	**Spare 23**
	22	**Spare 22**
	21	**Spare 21**
	20	**Spare 20**
	19	**Spare 19**
	18	**Spare 18**
	17	**Spare 17**
	16	**Spare 16**
	15	**Spare 15**
	14	**Spare 14**
	13	**Spare 13**
	12	**Spare 12**
	11	**Spare 11**

<table>
<tr><th colspan="3">DSSM</th></tr>
<tr><td></td><td>10</td><td>Spare 10</td></tr>
<tr><td></td><td>9</td><td>Spare 9</td></tr>
<tr><td></td><td>8</td><td>Spare 8</td></tr>
<tr><td></td><td>7</td><td>Spare 7</td></tr>
<tr><td></td><td>6</td><td>Spare 6</td></tr>
<tr><td></td><td>5</td><td>Spare 5</td></tr>
<tr><td></td><td>4</td><td>Spare 4</td></tr>
<tr><td></td><td>3</td><td>Spare 3</td></tr>
<tr><td></td><td>2</td><td>LCPLL Unavail
This bit specifies the availability of some LCPLL output frequencies.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Available</td><td>LCPLL available</td></tr>
<tr><td>1b</td><td>Not available</td><td>LCPLL not available</td></tr>
</table></td></tr>
<tr><td></td><td>1</td><td>Spare 1</td></tr>
<tr><td></td><td>0</td><td>DisplayPort A Present
This bit specifies whether the port was present during initalization. This strap state can also be read in the DDI_BUF_CTL_A 0x64000 register bit 0.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Not Present</td><td>Port not present</td></tr>
<tr><td>1b</td><td>Present</td><td>Port present</td></tr>
</table></td></tr>
</table>

DX9 Constants Not Consumed By RCS

DX9CONST_PRODUCE_COUNT - DX9 Constants Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02484h

This register keeps track of the outstanding DX9 Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**DX9 Constants Produce Count** This register keeps track of the outstanding DX9 Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

DX9 Constants Prsed By RCS

DX9CONST_PARSE_COUNT - DX9 Constants Prsed By RCS

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02494h

This register keeps track of the DX9 Constant commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has DX9 Pool Alloc Valid. DX9 parse count should be less then equal to the DX9 produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. DX9 Parse count is subtracted from the DX9 Produce Count upon parsing 3D_PRIMITIVE command. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**DX9 Constants Produce Count** This register keeps track of the DX9 Constant commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has DX9 Pool Alloc Valid. DX9 parse count should be less then equal to the DX9 produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. DX9 Parse count is subtracted from the DX9 Produce Count upon parsing 3D_PRIMITIVE command.

ECO reg 1

ECOREG1 - ECO reg 1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00FF8h-00FFBh	
DWord	**Bit**	**Description**
0	31	**Lock Bit** Access: R/W Lock Lock bit for this register
	30:0	**Reserved**

ECO Reserved

ECORESRV - ECO Reserved		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000 [KBL]	
Size (in bits):	32	
Address:	09898h	
ECO Reserved bits		
DWord	**Bit**	**Description**
0	31:16	**ECO Reserved Bits** Access: R/WC
	15:0	**Ita p-value config** Access: R/WC

Element Descriptor Register

ELEM_DESCRIPTOR - Element Descriptor Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000
Access:	RO
Size (in bits):	64
Address:	04500h
Name:	BCS Element Descriptor Register
ShortName:	BCS_ELEM_DESCRIPTOR
Address:	04400h
Name:	RCS Element Descriptor Register
ShortName:	RCS_ELEM_DESCRIPTOR
Address:	04440h
Name:	VCS Element Descriptor Register
ShortName:	VCS_ELEM_DESCRIPTOR
Address:	044C0h
Name:	VECS Element Descriptor Register
ShortName:	VECS_ELEM_DESCRIPTOR

Element Information: The register is populated by command streamer and consumed by GAM

DWord	Bit	Description
0	63:32	**Context ID** Context identification number assigned to separate this context from others. Context IDs needs to be recycled in such a way that there could not be two active context with the same ID. This is a unique identification number by which a context is identified and referenced
	31:12	**LRCA** Command Streamer Only
	11:9	**Function Number** GFX device is considered to be on Bus0 with device number of 2. Function number is normally assigned as "0" however for gfx virtualization; there would be different function numbers which needs to be attached to context. Not used in Gen8.
	8	**Privileged Context / GGTT vs PPGTT mode** In Legacy Context: Defines the page tables to be used. This is how page walker come to know PPGTT vs GGTT selection for the entire context. In Advanced Context: Defines the privilege level for the context

Value	Name	Description
0h	**[Default]**	Use Global GTT (In Legacy Context)

ELEM_DESCRIPTOR - Element Descriptor Register

		User Mode Context (In Advanced Context)
1h		Use Per-Process GTT (In Legacy Context) Supervisor Mode Context (In Advanced Context)

7:6 Fault Model

Value	Name	Description
00h	**[Default]**	Fault and Hang. Same mode as gen7.5
01h		Fault and Halt/Wait. Same mode as gen7.5
10h		Fault and Stream and Switch
11h		Reserved.

5 Deeper IA coherency Support

In Advanced Context: Defines the level of IA coherency

Value	Name	Description
0h	**[Default]**	IA coherency is provided at LLC level for all streams of GPU (i.e. gen7.5 like mode)
1h		IA coherency is provided at L3 level for EU data accesses of GPU

4 A and D Support / 32 and 64b Address Support

In Legacy Context: Defines 32b vs 64b (48b canonical) addressing format
In Advanced Context: Defines A/D bit support

Value	Name	Description
0h	**[Default]**	32b addressing format (In Legacy Context) A/D bit management in page tables is NOT supported (In Advanced Context)
1h		64b (48b canonical) addressing format (In Legacy Context) A/D bit management in page tables is supported (In Advanced Context)

3 Context Type: Legacy vs Advanced

Defines the context type.
Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected.

Value	Name	Description
0h	**[Default]**	Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models...). Note that advanced context is not bounded to GPGPU.
1h		Legacy Context: Defines the context as legacy mode which is similar to prior generations of gen8.

2 FR

Command Streamer Specific

ELEM_DESCRIPTOR - Element Descriptor Register

	1	**Scheduling Mode** Value / Name / Description 0h / **[Default]** / Indicates execlist mode of scheduling. 1h / / Indicates Ring Buffer mode of scheduling.
	0	**Valid** Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it will continue but flag an error.

EMRR Mask LSB

EMRRMASK_LSB - EMRR Mask LSB		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	09208h	
EMRR Mask Value		
DWord	**Bit**	**Description**
0	31:12	**EMRR_MASK_LSB** Access: RO EMRR MASK VALUE.
	11	**EMRR ENABLE** Access: RO EMRR Enable.
	10	**EMRR LOCK** Access: RO EMRR LOCK bit.
	9:0	**Spares** Access: RO

EMRR Mask MSB

EMRRMASK_MSB - EMRR Mask MSB		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0920Ch	
EMRR Mask Value		
DWord	**Bit**	**Description**
0	31:7	**Spares** Access: RO
	6:0	**EMRR_MASK_MSB** Access: RO EMRR MASK VALUE.

Error Identity Register

EIR - Error Identity Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Address:	020B0h-020B3h
Name:	Error Identity Register
ShortName:	EIR_RCSUNIT
Address:	120B0h-120B3h
Name:	Error Identity Register
ShortName:	EIR_VCSUNIT0
Address:	1A0B0h-1A0B3h
Name:	Error Identity Register
ShortName:	EIR_VECSUNIT
Address:	1C0B0h-1C0B3h
Name:	Error Identity Register
ShortName:	EIR_VCSUNIT1
Address:	220B0h-220B3h
Name:	Error Identity Register
ShortName:	EIR_BCSUNIT

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a 1 to the appropriate bit(s)), except for the unrecoverable bits described.)

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask
	15:0	**Error Identity Bits** Format: Array of Error condition bits See the table titled Hardware-Detected Error Bits. This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See Table Table 3-3. Hardware-Detected Error Bits). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

EIR - Error Identity Register

Reserved bits are RO.

Value	Name
1h	Error occurred

Programming Notes
Writing a 1 to a set bit will cause that error condition to be cleared. However, neither the Page Table Error bit (Bit 4) nor the Instruction Error bit (Bit 0) can be cleared except by reset (i.e., it is a fatal error).

Error Mask Register

EMR - Error Mask Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32

Address:	020B4h-020B7h
Name:	Error Mask Register
ShortName:	EMR_RCSUNIT

Address:	120B4h-120B7h
Name:	Error Mask Register
ShortName:	EMR_VCSUNIT0

Address:	1A0B4h-1A0B7h
Name:	Error Mask Register
ShortName:	EMR_VECSUNIT

Address:	1C0B4h-1C0B7h
Name:	Error Mask Register
ShortName:	EMR_VCSUNIT1

Address:	220B4h-220B7h
Name:	Error Mask Register
ShortName:	EMR_BCSUNIT

The EMR register is used by software to control which Error Status Register bits are masked or unmasked. Unmasked bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. Masked bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts. Reserved bits are RO.

DWord	Bit	Description
0	31:8	**Reserved** Default Value: FFFFFFh Format: Must Be One **Programming Notes** These bits are not implemented in HW and must be set to '1'
	7:0	**Error Mask Bits** Format: Array of error condition mask bits See the table titled Hardware-Detected Error Bits.

EMR - Error Mask Register

This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description
FFh	**[Default]**	
0h	Not Masked	Will be reported in the EIR
1h	Masked	Will not be reported in the EIR

Error Reporting Register

ERR - Error Reporting Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B42Ch	
DWord	**Bit**	**Description**
0	31:5	**Reserved** Access: RO Reserved.
	4	**First Content Buffer Ready 0** Access: R/W First Content Buffer Ready 0 (FRSNTBFR0). First Content Buffer Ready: This bit gets set by the HW when the buffer is completely filled up and cleared by the driver when the contents of this buffer are copied out of memory. Is set by lpfc_lpconf_buffer0_ready (pulse). lpconf_lpfc_buffer0_ready (static signal to lpfc).
	3	**Second Buffer ready slice 0** Access: R/W Second Content Buffer Ready slice 0 (SCNBFR0). Second Content Buffer Ready: This bit gets set by the HW when the buffer is completely filled up and cleared by the driver when the contents of this buffer are copied out of memory. Is set by lpfc_lpconf_buffer1_ready (pulse). lpconf_lpfc_buffer1_ready (static signal to lpfc).
	2	**Write Expire Error Slice 0** Access: R/W Write Expired Error slice 0 (WEERR0). Write Expired Error: If DMA controller could not get a chance to push the write of 64Bytes to LTISEQ and data gets clobbered with the new expiration of the save timer, this error bit is set to indicate something went wrong. Signal -lpfc_lpconf_wrexp_error.
	1	**Buffer full Error Slice 0** Access: R/W Buffer full Error Slice 0 (BFFLERR0). Set by lpfc_lpconf_error_buffer_full. When all buffers are full lpfc sets this bit or if only 1 buffer is enabled then lpfc sets this bit when the buffer is full.

ERR - Error Reporting Register

	0	**Reserved** Access: RO Reserved.

Error Status Register

ESR - Error Status Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32

Address:	020B8h-020BBh
Name:	Error Status Register
ShortName:	ESR_RCSUNIT

Address:	120B8h-120BBh
Name:	Error Status Register
ShortName:	ESR_VCSUNIT0

Address:	1A0B8h-1A0BBh
Name:	Error Status Register
ShortName:	ESR_VECSUNIT

Address:	1C0B8h-1C0BBh
Name:	Error Status Register
ShortName:	ESR_VCSUNIT1

Address:	220B8h-220BBh
Name:	Error Status Register
ShortName:	ESR_BCSUNIT

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition persistent). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Error Status Bits** Format: Array of error condition bits See the table titled Hardware-Detected Error Bits. This register contains the non-persistent values of all hardware-detected error condition bits. Value: 1h — Name: Error Condition Detected

EU_GRF_CLEAR

EU_GRF_CLEAR - EU_GRF_CLEAR			
Register Space:	MMIO: 0/2/0		
Source:	BSpec		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0E550h		
Name:	EU_GRF_CLEAR		
ShortName:	EU_GRF_CLEAR		
This is a basic register template			
DWord	**Bit**	**Description**	
0	31:0	**GRF_CLEAR**	
		Default Value:	0000000000000000b
		Access:	RO

EU_STALL PER SUBSLICE

EUMETRICS_EVENT0 - EU_STALL PER SUBSLICE	
Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00D8Ch

This register mirrors an accumulating count for EU Metric Event0.
It is enabled by configuration bits in GPMunit and SPMunits.
Note that count is never cleared and delta should be calculated by sampling the initial register value at the start and subtracting that value off from the final sample value.

DWord	Bit	Description
0	31:0	**EU Metric Event Count** Access: RO

EU Mask Programming

TD_PM_MODE_EUCOUNT - EU Mask Programming

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4F8h
Name:	EU Mask Programming Slice 0
ShortName:	TD_PM_MODE_EUCOUNT_S0
Address:	0E5F8h
Name:	EU Mask Programming Slice 1
ShortName:	TD_PM_MODE_EUCOUNT_S1
Address:	0E6F8h
Name:	EU Mask Programming Slice 2
ShortName:	TD_PM_MODE_EUCOUNT_S2

DWord	Bit	Description
0	31	**SubSlice 3 EU 7 Enable** Format: Enable Value / Name: 0 – Enabled **[Default]** 1 – Disabled
	30	**SubSlice 3 EU 6 Enable** Format: Enable Value / Name: 0 – Enabled **[Default]** 1 – Disabled
	29	**SubSlice 3 EU 5 Enable** Format: Enable Value / Name: 0 – Enabled **[Default]** 1 – Disabled

TD_PM_MODE_EUCOUNT - EU Mask Programming

	28	**SubSlice 3 EU 4 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	27	**SubSlice 3 EU 3 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	26	**SubSlice 3 EU 2 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	25	**SubSlice 3 EU 1 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	24	**SubSlice 3 EU 0 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	23	**SubSlice 2 EU 7 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled

TD_PM_MODE_EUCOUNT - EU Mask Programming

	22	**SubSlice 2 EU 6 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	21	**SubSlice 2 EU 5 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	20	**SubSlice 2 EU 4 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	19	**SubSlice 2 EU 3 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	18	**SubSlice 2 EU 2 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	17	**SubSlice 2 EU 1 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled

TD_PM_MODE_EUCOUNT - EU Mask Programming

	16	**SubSlice 2 EU 0 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled
	15	**SubSlice 1 EU 7 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled
	14	**SubSlice 1 EU 6 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled
	13	**SubSlice 1 EU 5 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled
	12	**SubSlice 1 EU 4 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled
	11	**SubSlice 1 EU 3 Enable** Format: Enable Value / Name 0: Enabled **[Default]** 1: Disabled

TD_PM_MODE_EUCOUNT - EU Mask Programming

	10	**SubSlice 1 EU 2 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	9	**SubSlice 1 EU 1 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	8	**SubSlice 1 EU 0 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	7	**SubSlice 0 EU 7 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	6	**SubSlice 0 EU 6 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled
	5	**SubSlice 0 EU 5 Enable** Format: Enable Value / Name: 0 — Enabled **[Default]** 1 — Disabled

TD_PM_MODE_EUCOUNT - EU Mask Programming

	4	**SubSlice 0 EU 4 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	3	**SubSlice 0 EU 3 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	2	**SubSlice 0 EU 2 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	1	**SubSlice 0 EU 1 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled
	0	**SubSlice 0 EU 0 Enable** Format: Enable Value / Name 0 — Enabled **[Default]** 1 — Disabled

EU NOT IDLE PER SUBSLICE

EUMETRICS_EVENT4 - EU NOT IDLE PER SUBSLICE		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00D9Ch	
This register mirrors an accumulating count for EU Metric Event4. It is enabled by configuration bits in GPMunit and SPMunits. Note that count is never cleared and delta should be calculated by sampling the initial register value at the start and subtracting that value off from the final sample value.		
DWord	**Bit**	**Description**
0	31:0	**EU Metric Event Count** Access: RO

EU PAIR 0 PFET control register with lock

PFETCTL - EU PAIR 0 PFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0001000A
Size (in bits):	32
Address:	24608h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of EU PAIR 0 PFETCTL register are R/W 1 = All bits of EU PAIR 0 PFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:21	**Reserved** Access: RO Reserved
	20	**Reserved**
	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered up Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good.** Default Value: 001b Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: 40ns 3'b001: 80ns 3'b010: 160ns 3'b011: 320ns 3'b100: 640ns 3'b101: 1280ns 3'b110: 2560ns 3'b111: 5120ns

PFETCTL - EU PAIR 0 PFET control register with lock

	15:13	**Time period last primay pfet strobe to secondary pfet strobe** Access: R/W Lock Time period last primay pfet strobe to secondary pfet strobe 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	12:10	**Time period b/w two adjacent strobes** Access: R/W Lock Time period b/w two adjacent strobes to the primary FETs 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	9:7	**FET setup margin from enable to strobe** Access: R/W Lock Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	6:0	**Number of flops to enable primary FETs** Default Value: 0001010b Access: R/W Lock Number of flops to enable primary FETs. For a setting of N there will be N+1 total strobes generated 7'b0000000: 10 Flops to be strobed 7'b0000001: 11 Flops to be strobed 7'b0000010: 12 Flops to be strobed 7'b0001111: 26 Flops to be strobed

EU PAIR 0 PGFET control register with lock

PFETCTL - EU PAIR 0 PGFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0001000A
Size (in bits):	32
Address:	24608h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of EU PAIR 0 PGFETCTL register are R/W 1 = All bits of EU PAIR 0 PGFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:21	**Reserved** Access: RO Reserved
	20	**Reserved**
	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered Up Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good.** Default Value: 001b Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: 40ns 3'b001: 80ns 3'b010: 160ns 3'b011: 320ns 3'b100: 640ns 3'b101: 1280ns 3'b110: 2560ns 3'b111: 5120ns

PFETCTL - EU PAIR 0 PGFET control register with lock

	15:13	**Time period last primay pfet strobe to secondary pfet strobe** Access: R/W Lock Time period last primay pfet strobe to secondary pfet strobe 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	12:10	**Time period b/w two adjacent strobes** Access: R/W Lock Time period b/w two adjacent strobes to the primary FETs 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	9:7	**FET setup margin from enable to strobe** Access: R/W Lock Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	6:0	**Number of flops to enable primary FETs** Default Value: 0001010b Access: R/W Lock Number of flops to enable primary FETs. For a setting of N there will be N+1 total strobes generated 7'b0000000: 10 Flops to be strobed 7'b0000001: 11 Flops to be strobed 7'b0000010: 12 Flops to be strobed 7'b0001111: 26 Flops to be strobed

EU PAIR 0 Power Context Save request

PGCTXREQ - EU PAIR 0 Power Context Save request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24604h	
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask bots for lower 16 bits
	15:10	**Reserved** Access: RO Reserved
	9	**Power context save request** Access: R/W Set Power Context Save Request 1'b0 : Power context save is not being requested <default> 1'b1 : Power context save is being requested CPUnit self-clears this bit upon sampling.
	8:0	**Power Context Save request crdit count** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

EU PAIR 0 Power Down FSM control register with lock

POWERDNFSMCTL - EU PAIR 0 Power Down FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	24610h

DWord	Bit	Description
0	31	**power down control Lock** Access: R/W Lock 0 = Bits of EU PAIR 0 POWERDNFSMCTL register are R/W 1 = All bits of EU PAIR 0 POWERDNFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:13	**Reserved** Access: RO Reserved
	12	**Leave firewall disabled** Access: R/W Lock When this bit is set, SPC will not firewall the gated domain for a power down flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, firewall gated domain to ungated domain crossing during power down flows 1 = Leave firewall disabled, that is, dont firewall the gated domain, but complete logical flow
	11	**Leave reset de-asserted** Access: R/W Lock When this bit is set, SPC will not assert reset for power off flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, assert resets during power down flows 1 = Leave reset de-asserted mode, that is, dont assert reset, but complete logical flow

POWERDNFSMCTL - EU PAIR 0 Power Down FSM control register with lock

10	**Leave CLKs ON** Access: R/W Lock When this bit is set, SPC will not gate clks for power off flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, gate clocks during power down flows 1 = Leave CLKS ON mode, that is, dont clock gate, but complete logical flow
9	**Leave FET On** Access: R/W Lock When this bit is set, SPC will not turn off the PFET eventhough it will complete the flow with PM. Encodings: 0 = Default mode, that is, power off fets during power down flows 1 = Leave ON mode, that is, dont power off pfet, but complete logical flow **Programming Notes** This bit should be programmed before the powerup sequence is initiated for SSM.
8:6	**Power Down state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Gate Clocks
5:3	**Power Down state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default :Firewall ON

POWERDNFSMCTL - EU PAIR 0 Power Down FSM control register with lock

	2:0	**Power Down state 1** Default Value: 000b Access: R/W Lock This will be the 1st state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Assert Reset

EU PAIR 0 Power Gate Control Request

PGCTLREQ - EU PAIR 0 Power Gate Control Request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24600h	
Clock Gating Messages Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask - To write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:2	**Reserved** Access: RO Reserved
	1	**CLK RST FWE Request** Access: R/W EU PAIR 0 CLK RST FWE request: '0' : Initiate power down sequence (clk/rst/fwe) '1' : Initiate power up sequence (clk/rst/fwe)
	0	**Reserved**

EU PAIR 0 Power on FSM control register with lock

POWERUPFSMCTL - EU PAIR 0 Power on FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	2460Ch

DWord	Bit	Description
0	31	**power up control Lock** Access: R/W Lock 0 = Bits of EU PAIR 0 POWERUPFSMCTL register are R/W 1 = All bits of EU PAIR 0 POWERUPFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:9	**Reserved** Access: RO Reserved
	8:6	**Power UP state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - De-assert resets 3'b000: 10ns (or 1 bclk)
	5:3	**Power UP state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets

POWERUPFSMCTL - EU PAIR 0 Power on FSM control register with lock

		1xx = Rsvd for future Default - Firewall OFF
	2:0	**Power UP state 1** Default Value: 000b Access: R/W Lock This will be the 1st state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - Clock Ungate

EU PAIR 1 PFET control register with lock

PFETCTL - EU PAIR 1 PFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0001000A [KBL]
Size (in bits):	32
Address:	24688h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of EU PAIR 1 PFETCTL register are R/W 1 = All bits of EU PAIR 1 PFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:21	**Reserved** Access: RO Reserved
	20	**Power Well Status** Access: R/WC 0 = Well is powered Down 1 = Well is powered up Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered up Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good.** Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: 40ns 3'b001: 80ns 3'b010: 160ns 3'b011: 320ns

PFETCTL - EU PAIR 1 PFET control register with lock

3'b100: 640ns
3'b101: 1280ns
3'b110: 2560ns
3'b111: 5120ns

Value	Name
001b	**[Default]**

15:13 **Time period last primay pfet strobe to secondary pfet strobe**

Access:	R/W Lock

Time period last primay pfet strobe to secondary pfet strobe
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

12:10 **Time period b/w two adjacent strobes**

Access:	R/W Lock

Time period b/w two adjacent strobes to the primary FETs
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

9:7 **FET setup margin from enable to strobe**

Access:	R/W Lock

Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

6:0 **Number of flops to enable primary FETs**

Default Value:	0001010b
Access:	R/W Lock

Number of flops to enable primary FETs. For a setting of N there will be N+1 total strobes generated
7'b0000000: 10 Flops to be strobed
7'b0000001: 11 Flops to be strobed
7'b0000010: 12 Flops to be strobed
7'b0001111: 26 Flops to be strobed

EU PAIR 1 Power Context Save request

PGCTXREQ - EU PAIR 1 Power Context Save request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24684h	
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask bots for lower 16 bits
	15:10	**Reserved** Access: RO Reserved
	9	**Power context save request** Access: R/W Set Power Context Save Request 1'b0 : Power context save is not being requested <default> 1'b1 : Power context save is being requested CPUnitself-clears this bit upon sampling.
	8:0	**Power Context Save request crdit count** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

EU PAIR 1 Power Down FSM control register with lock

POWERDNFSMCTL - EU PAIR 1 Power Down FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	24690h

DWord	Bit	Description
0	31	**power down control Lock** Access: R/W Lock 0 = Bits of EU PAIR 1 POWERDNFSMCTL register are R/W 1 = All bits of EU PAIR 1 POWERDNFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:13	**Reserved** Access: RO Reserved
	12	**Leave firewall disabled** Access: R/W Lock When This bit is set SPC will not firewall the gated domain for a power down flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e firewall gated domain to ungated domain crossing during power down flows 1 = Leave firewall disabled, i.e dont firewall the gated domain, but complete logical flow
	11	**Leave reset de-asserted** Access: R/W Lock When This bit is set SPC will not assert reset for power off flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e assert resets during power down flows 1 = Leave reset de-asserted mode, i.e dont assert reset, but complete logical flow

POWERDNFSMCTL - EU PAIR 1 Power Down FSM control register with lock

	10	**Leave CLKs ON** Access: R/W Lock When This bit is set SPC will not gate clks for power off flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e gate clocks during power down flows 1 = Leave CLKS ON mode, i.e dont clock gate, but complete logical flow
	9	**Leave FET On** Access: R/W Lock When This bit is set SPC will not turn off the PFET eventhough it will complete the flow with PM Encodings: 0 = Default mode, i.e power off fets during power down flows 1 = Leave ON mode, i.e dont power off pfet, but complete logical flow Programming note : This bit should be programmed before the powerup sequence is initiated for EUP1
	8:6	**Power Down state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Gate Clocks
	5:3	**Power Down state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default :Firewall ON

POWERDNFSMCTL - EU PAIR 1 Power Down FSM control register with lock

	2:0	**Power Down state 1** Default Value: 000b Access: R/W Lock This will be the 1st state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Assert Reset

EU PAIR 1 Power Gate Control Request

PGCTLREQ - EU PAIR 1 Power Gate Control Request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24680h	
Clock Gating Messages Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:2	**Reserved** Access: RO Reserved
	1	**CLK RST FWE Request** Access: R/W EU PAIR 1 CLK RST FWE request: '0' : Initiate power down sequence (clk/rst/fwe) '1' : Initiate power up sequence (clk/rst/fwe)
	0	**Power Gate Request** Access: R/W EU PAIR 1 power well request: '0' : Initiate Power Down request '1' : Initiate Power UP req

EU PAIR 1 Power on FSM control register with lock

POWERUPFSMCTL - EU PAIR 1 Power on FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	2468Ch

DWord	Bit	Description
0	31	**power up control Lock** Access: R/W Lock 0 = Bits of EU PAIR 1 POWERUPFSMCTL register are R/W 1 = All bits of EU PAIR 1 POWERUPFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:9	**Reserved** Access: RO Reserved
	8:6	**Power UP state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - De-assert resets 3'b000: 10ns (or 1 bclk)
	5:3	**Power UP state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF

POWERUPFSMCTL - EU PAIR 1 Power on FSM control register with lock

		010 = De-assert resets 1xx = Rsvd for future Default - Firewall OFF
	2:0	**Power UP state 1** Default Value: 000b Access: R/W Lock This will be the 1st state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - Clock Ungate

EU PAIR 2 PGFET control register with lock

PFETCTL - EU PAIR 2 PGFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0001000A [KBL]
Size (in bits):	32
Address:	24708h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of EU PAIR 2 PGFETCTL register are R/W 1 = All bits of EU PAIR 2 PGFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:21	**Reserved** Access: RO Reserved
	20	**Power Well Status** Access: R/WC 0 = Well is powered Down 1 = Well is powered Up Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered Up Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good** Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: 40ns 3'b001: 80ns 3'b010: 160ns 3'b011: 320ns

PFETCTL - EU PAIR 2 PGFET control register with lock

3'b100: 640ns
3'b101: 1280ns
3'b110: 2560ns
3'b111: 5120ns

Value	Name
001b	**[Default]**

15:13 **Time period last primay pfet strobe to secondary pfet strobe**

Access:	R/W Lock

Time period last primay pfet strobe to secondary pfet strobe
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

12:10 **Time period b/w two adjacent strobes**

Access:	R/W Lock

Time period b/w two adjacent strobes to the primary FETs
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

9:7 **FET setup margin from enable to strobe**

Access:	R/W Lock

Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop
3'b000: 10ns (or 1 bclk)
3'b001: 20ns (or 2 bclk)
3'b010: 30ns (or 3 bclk)
3'b111: 80ns (or 8 bclk)

6:0 **Number of flops to enable primary FETs**

Default Value:	0001010b
Access:	R/W Lock

Number of flops to enable primary FETs. For a setting of N, there will be N+1 total strobes generated.
7'b0000000: 10 Flops to be strobed
7'b0000001: 11 Flops to be strobed
7'b0000010: 12 Flops to be strobed
7'b0001111: 26 Flops to be strobed

EU PAIR 2 Power Context Save request

PGCTXREQ - EU PAIR 2 Power Context Save request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24704h	
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask bits for lower 16 bits
	15:10	**Reserved** Access: RO Reserved
	9	**Power context save request** Access: R/W Set Power Context Save Request 1'b0 : Power context save is not being requested <default> 1'b1 : Power context save is being requested CPUnit self-clears this bit upon sampling.
	8:0	**Power Context Save request crdit count** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32 bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

EU PAIR 2 power Down FSM control register with lock

POWERDNFSMCTL - EU PAIR 2 power Down FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	24710h

DWord	Bit	Description
0	31	**power Down control Lock** Access: R/W Lock 0 = Bits of EU PAIR 2 POWERDNFSMCTL register are R/W 1 = All bits of EU PAIR 2 POWERDNFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:13	**Reserved** Access: RO Reserved
	12	**Leave firewall disabled** Access: R/W Lock When this bit is set, SPC will not firewall the gated domain for a power Down flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, firewall gated domain to ungated domain crossing during power Down flows 1 = Leave firewall disabled, that is, don't firewall the gated domain, but complete logical flow
	11	**Leave reset de-asserted** Access: R/W Lock When this bit is set SPC will not assert reset for power off flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, assert resets during power Down flows 1 = Leave reset de-asserted mode, that is, don't assert reset, but complete logical flow

POWERDNFSMCTL - EU PAIR 2 power Down FSM control register with lock

	10	**Leave CLKs ON** Access: R/W Lock When this bit is set SPC will not gate clks for power off flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, gate clocks during power Down flows 1 = Leave CLKS ON mode, that is, don't clock gate, but complete logical flow
	9	**Leave FET On** Access: R/W Lock When this bit is set SPC will not turn off the PFET eventhough it will complete the flow with PM. Encodings: 0 = Default mode, that is, power off fets during power Down flows 1 = Leave ON mode, that is, don't power off pfet, but complete logical flow **Programming Notes** This bit should be programmed before the powerup sequence is initiated for EUP2.
	8:6	**power Down state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state before power is turned OFF in the well. Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future **Default - Gate Clocks**
	5:3	**power Down state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state before power is turned OFF in the well. Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future **Default - Firewall ON**

POWERDNFSMCTL - EU PAIR 2 power Down FSM control register with lock

	2:0	**power Down state 1** Default Value: 000b Access: R/W Lock This will be the 1st state before power is turned OFF in the well. Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future **Default - Assert Reset**

EU PAIR 2 Power Gate Control Request

PGCTLREQ - EU PAIR 2 Power Gate Control Request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24700h	
Clock Gating Messages Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask - To write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000.
	15:2	**Reserved** Access: RO Reserved
	1	**CLK RST FWE Request** Access: R/W EU PAIR 2 CLK RST FWE request: '0' : Initiate power Down sequence (clk/rst/fwe) '1' : Initiate power Up sequence (clk/rst/fwe)
	0	**Power Gate Request** Access: R/W EU PAIR 2 power well request: '0' : Initiate power Down request '1' : Initiate power Up req

EU PAIR 2 Power on FSM control register with lock

POWERUPFSMCTL - EU PAIR 2 Power on FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	2470Ch

DWord	Bit	Description
0	31	**power Up control Lock** Access: R/W Lock 0 = Bits of EU PAIR 2 POWERUPFSMCTL register are R/W 1 = All bits of EU PAIR 2 POWERUPFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:9	**Reserved** Access: RO Reserved
	8:6	**power Up state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future **Default - De-assert resets** 3'b000: 10ns (or 1 bclk)
	5:3	**power Up state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state after power is turned ON in the well. Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future **Default - Firewall OFF**

POWERUPFSMCTL - EU PAIR 2 Power on FSM control register with lock

	2:0	**power Up state 1** Default Value: 000b Access: R/W Lock This will be the 1st state after power is turned ON in the well. Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future **Default - Clock Ungate**

EU PAIR 3 PFET control register with lock

PFETCTL - EU PAIR 3 PFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0001000A
Size (in bits):	32
Address:	24788h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of EU PAIR 3 PFETCTL register are R/W 1 = All bits of EU PAIR 3 PFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:21	**Reserved** Access: RO Reserved
	20	**Power Well Status** Access: R/WC 0 = Well is powered Down 1 = Well is powered up Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered up Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good.** Default Value: 001b Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: 40ns 3'b001: 80ns

PFETCTL - EU PAIR 3 PFET control register with lock

		3'b010: 160ns 3'b011: 320ns 3'b100: 640ns 3'b101: 1280ns 3'b110: 2560ns 3'b111: 5120ns
	15:13	**Time period last primay pfet strobe to secondary pfet strobe** Access: R/W Lock Time period last primay pfet strobe to secondary pfet strobe 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	12:10	**Time period b/w two adjacent strobes** Access: R/W Lock Time period b/w two adjacent strobes to the primary FETs 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	9:7	**FET setup margin from enable to strobe** Access: R/W Lock Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop 3'b000: 10ns (or 1 bclk) 3'b001: 20ns (or 2 bclk) 3'b010: 30ns (or 3 bclk) 3'b111: 80ns (or 8 bclk)
	6:0	**Number of flops to enable primary FETs** Default Value: 0001010b Access: R/W Lock Number of flops to enable primary FETs. For a setting of N there will be N+1 total strobes generated 7'b0000000: 10 Flops to be strobed 7'b0000001: 11 Flops to be strobed 7'b0000010: 12 Flops to be strobed 7'b0001111: 26 Flops to be strobed

EU PAIR 3 Power Context Save request

PGCTXREQ - EU PAIR 3 Power Context Save request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24784h	
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask bots for lower 16 bits
	15:10	**Reserved** Access: RO Reserved
	9	**Power context save request** Access: R/W Set Power Context Save Request 1'b0 : Power context save is not being requested <default> 1'b1 : Power context save is being requested CPUnit self-clears this bit upon sampling.
	8:0	**Power Context Save request crdit count** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

EU PAIR 3 Power Down FSM control register with lock

POWERDNFSMCTL - EU PAIR 3 Power Down FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	24790h

DWord	Bit	Description
0	31	**power down control Lock** Access: R/W Lock 0 = Bits of EU PAIR 3 POWERDNFSMCTL register are R/W 1 = All bits of EU PAIR 3 POWERDNFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:13	**Reserved** Access: RO Reserved
	12	**Leave firewall disabled** Access: R/W Lock When This bit is set SPC will not firewall the gated domain for a power down flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e firewall gated domain to ungated domain crossing during power down flows 1 = Leave firewall disabled, i.e dont firewall the gated domain, but complete logical flow
	11	**Leave reset de-asserted** Access: R/W Lock When This bit is set SPC will not assert reset for power off flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e assert resets during power down flows 1 = Leave reset de-asserted mode, i.e dont assert reset, but complete logical flow

POWERDNFSMCTL - EU PAIR 3 Power Down FSM control register with lock

	10	**Leave CLKs ON** Access: R/W Lock When This bit is set SPC will not gate clks for power off flow. But it will pretend to complete the flow with PM Encodings: 0 = Default mode, i.e gate clocks during power down flows 1 = Leave CLKS ON mode, i.e dont clock gate, but complete logical flow
	9	**Leave FET On** Access: R/W Lock When This bit is set SPC will not turn off the PFET eventhough it will complete the flow with PM Encodings: 0 = Default mode, i.e power off fets during power down flows 1 = Leave ON mode, i.e dont power off pfet, but complete logical flow **Programming Notes** This bit should be programmed before the powerup sequence is initiated for EUP3.
	8:6	**Power Down state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Gate Clocks
	5:3	**Power Down state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default :Firewall ON

POWERDNFSMCTL - EU PAIR 3 Power Down FSM control register with lock

	2:0	**Power Down state 1** Default Value: 000b Access: R/W Lock This will be the 1st state before power is turned OFF in the well Encodings: 000 = Assert Reset 001 = Firewall ON 010 = Gate clocks 1xx = Rsvd for future Default : Assert Reset

EU PAIR 3 Power Gate Control Request

PGCTLREQ - EU PAIR 3 Power Gate Control Request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24780h	
Clock Gating Messages Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask - To write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:2	**Reserved** Access: RO Reserved
	1	**CLK RST FWE Request** Access: R/W EU PAIR 3 CLK RST FWE request: '0' : Initiate power down sequence (clk/rst/fwe) '1' : Initiate power up sequence (clk/rst/fwe)
	0	**Power Gate Request** Access: R/W EU PAIR 3 power well request: '0' : Initiate Power Down request '1' : Initiate Power UP req

EU PAIR 3 Power on FSM control register with lock

POWERUPFSMCTL - EU PAIR 3 Power on FSM control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000088
Size (in bits):	32
Address:	2478Ch

DWord	Bit	Description
0	31	**power up control Lock** Access: R/W Lock 0 = Bits of EU PAIR 3 POWERUPFSMCTL register are R/W 1 = All bits of EU PAIR 3 POWERUPFSMCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:9	**Reserved** Access: RO Reserved
	8:6	**Power UP state 3** Default Value: 010b Access: R/W Lock This will be the 3rd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - De-assert resets 3'b000: 10ns (or 1 bclk)
	5:3	**Power UP state 2** Default Value: 001b Access: R/W Lock This will be the 2nd state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF

POWERUPFSMCTL - EU PAIR 3 Power on FSM control register with lock

		010 = De-assert resets 1xx = Rsvd for future Default - Firewall OFF
	2:0	**Power UP state 1** Default Value: 000b Access: R/W Lock This will be the 1st state after power is turned ON in the well Encodings: 000 = Clock Ungate 001 = Firewall OFF 010 = De-assert resets 1xx = Rsvd for future Default - Clock Ungate

Event selection and base counters

LPFCREG2 - Event selection and base counters		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B00Ch	
DWord	**Bit**	**Description**
0	31:24	**Counter 7 client** Access: R/W Incf_lpfc_cnt7_client[7:0]. Client Encoding (hex): GAFS Rd 00 GAFS Wr 01 HDC0 Data Rd 02 HDC0 Const Rd 03 HDC0 URB Rd 04 HDC0 Data Wr 05 HDC0 URB Wr 06 HDC1 Data Rd 07 HDC1 Const Rd 08 HDC1 URB Rd 09 HDC1 Data Wr 0A HDC1 URB Wr 0B TDL0 Rd 0C TDL1 Rd 0D Tex0 Rd 0E Tex1 Rd 0F Tex2 Rd (reserved) 10 Tex3 Rd (reserved) 11 SBE Rd 12 IC0 Rd 13 IC1 Rd 14 SARB Rd 15 Aggregated Tex 16 SLM0 Rd 17 SLM1 Rd 18 SLM0 Wr 19 SLM1 Wr 1A SLM0 Atomics 1B SLM1 Atomics 1C Reserved 1D

LPFCREG2 - Event selection and base counters		
		Reserved 1E Reserved 1F FF Stalls 20 HDC Stalls 21 TDL Stalls 22 Texture Stalls 23 IC Stalls 24 SBE Stalls 25 SLM Stalls 26 Bank0 Total Hits 40 Bank0 Total Cycles 41 Bank0 Total Rds 42 Bank0 Total Wrs 43 Bank0 FF Rds 44 Bank0 FF Wrs 45 Bank0 DC Rds 46 Bank0 DC Wrs 47 Bank0 DC Hits 48 rsvd 49 Bank0 Tex Rds 4A Bank0 Tex Hits 4B Bank0 IC Rds 4C Bank0 IC Hits 4D Reserved 4E Reserved 4F Bank1 Events 50-5F (except 59-reserved) Bank2 Events 60-6F(except 69-reserved) Bank3 Events 70-7F(except 79-reserved) MSC Rd 80 MSC Wr 81 STC Rd 82 STC Wr 83 Hiz Rd 84 Hiz Wr 85 RCZ Rd 86 RCZ Wr 87 RCC Rd 88 RCC Wr 89 LTCD0 Err Corr EE LTCD1 Err Corr EF LTCD2 Err Corr F0 LTCD3 Err Corr F1 LTCD0 Err UnCorr F2 LTCD1 Err UnCorr F3 LTCD2 Err UnCorr F4 LTCD3 Err UnCorr F5

LPFCREG2 - Event selection and base counters

		Counter#7 Client Selection: This field controls which client's request stream is observed in counter#7.
	23:16	**Counter 6 client** Access: R/W lncf_lpfc_cnt6_client[7:0]. Counter#6 Client Selection: This field controls which client's request stream is observed in counter#6.
	15:8	**Counter 5 client** Access: R/W lncf_lpfc_cnt5_client[7:0]. Counter#5 Client Selection: This field controls which client's request stream is observed in counter#5.
	7:0	**Counter 4 client** Access: R/W lncf_lpfc_cnt4_client[7:0]. Counter#4 Client Selection: This field controls which client's request stream is observed in counter#4.

Event Selection and Base Counters1

LPFCREG1 - Event Selection and Base Counters1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B010h	
DWord	**Bit**	**Description**
0	31:24	**Counter 3 client** Access: R/W lncf_lpfc_cnt3_client[7:0]. Counter#3 Client Selection: This field controls which client's request stream is observed in counter#3.
	23:16	**Counter 2 client** Access: R/W lncf_lpfc_cnt2_client[7:0]. Counter#2 Client Selection: This field controls which client's request stream is observed in counter#2.
	15:8	**Counter 1 Client** Access: R/W lncf_lpfc_cnt1_client[7:0]. Counter#1 Client Selection: This field controls which client's request stream is observed in counter#1.
	7:0	**Counter0 Client** Access: R/W lncf_lpfc_cnt0_client[7:0]. Counter#0 Client Selection: This field controls which client's request stream is observed in counter#0.

Execlist 0 Contents

EXECLIST0_CONTENTS - Execlist 0 Contents	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	R/W
Size (in bits):	128
Trusted Type:	1
Address:	02250h-0225Fh
Name:	Execlist 0 Contents
ShortName:	EXECLIST0_CONTENTS_RCSUNIT
Address:	12250h-1225Fh
Name:	Execlist 0 Contents
ShortName:	EXECLIST0_CONTENTS_VCSUNIT0
Address:	1A250h-1A25Fh
Name:	Execlist 0 Contents
ShortName:	EXECLIST0_CONTENTS_VECSUNIT
Address:	1C250h-1C25Fh
Name:	Execlist 0 Contents
ShortName:	EXECLIST0_CONTENTS_VCSUNIT1
Address:	22250h-2225Fh
Name:	Execlist 0 Contents
ShortName:	EXECLIST0_CONTENTS_BCSUNIT

Contents of the Execlist 0 in HW.

DWord	Bit	Description
0	31:0	**Element 0 Low DWord** Format: ContextDescriptorHigh
1	31:0	**Element 0 High DWord** Format: ContextDescriptorLow
2	31:0	**Element 1 Low DWord** Format: ContextDescriptorHigh
3	31:0	**Element 1 High DWord** Format: ContextDescriptorLow

Execlist 1 Contents

EXECLIST1_CONTENTS - Execlist 1 Contents	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	R/W
Size (in bits):	128
Trusted Type:	1
Address:	02260h-0226Fh
Name:	Execlist 1 Contents
ShortName:	EXECLIST1_CONTENTS_RCSUNIT
Address:	12260h-1226Fh
Name:	Execlist 1 Contents
ShortName:	EXECLIST1_CONTENTS_VCSUNIT0
Address:	1A260h-1A26Fh
Name:	Execlist 1 Contents
ShortName:	EXECLIST1_CONTENTS_VECSUNIT
Address:	1C260h-1C26Fh
Name:	Execlist 1 Contents
ShortName:	EXECLIST1_CONTENTS_VCSUNIT1
Address:	22260h-2226Fh
Name:	Execlist 1 Contents
ShortName:	EXECLIST1_CONTENTS_BCSUNIT

Contents of the Execlist 1 in HW.

DWord	Bit	Description
0	31:0	**Element 0 Low DWord** Format: ContextDescriptorHigh
1	31:0	**Element 0 High DWord** Format: ContextDescriptorLow
2	31:0	**Element 1 Low DWord** Format: ContextDescriptorHigh
3	31:0	**Element 1 High DWord** Format: ContextDescriptorLow

Execlist Status

EXECLIST_STATUS - Execlist Status	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000001, 0x00000000
Access:	RO
Size (in bits):	64
Address:	02234h-0223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_RCSUNIT
Address:	12234h-1223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VCSUNIT0
Address:	1A234h-1A23Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VECSUNIT
Address:	1C234h-1C23Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VCSUNIT1
Address:	22234h-2223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_BCSUNIT

This register contains the pointers and full indicator for the Execlist Queue and the context ID of the currently running context.
DefaultValue = UUUU UUU1h (4:0 default to 00001b, others UNDEFINED).

DWord	Bit	Description
0	63:32	**Current Context ID** Format: U32 Contains the context ID of the currently running context.
	31:30	**Reserved** Format: MBZ
	29:28	**Reserved** Format: MBZ

EXECLIST_STATUS - Execlist Status

27 Reserved

Format:	MBZ

26:19 Reserved

Format:	MBZ

18 Reserved

17 Reserved

16 Arbitration Enable

Format:	U1

This field reflects the Arbitration Flag set by the MI_ARB_ON_OFF command in Command Streamer.

15:14 Current Active Element Status

Format:	U2

Points at the element being executed in current Execlist (if there is one).

Value	Name
00b	No Active Element being executed
01b	Element0 of current execlist being executed
10b	Element1 of current execlist being executed
11b	Reserved

13:5 Last Context Switch Reason

Access:	R/W
Format:	U9

This field contains the switch reason for the last context to switch away, as captured in the Context Status Dword, bits 8:0.

Programming Notes
This field should not written by SW.

4 Execlist 0 Valid

Format:	Flag

This bit is set when the first DW for this Execlist port 0 is written through the submission port, and will not be cleared till the CSB is updated and the command stream is switching to the next execution list. If no execution list is pending, the transition of this bit from one to zero guarentees there will be no preemption on the next submission.

Value	Name
0	Invalid **[Default]**
1	Valid

EXECLIST_STATUS - Execlist Status

3 — Execlist 1 Valid

Format:	Flag

This bit is set when the first DW for this Execlist port 1 is written through the submission port, and will not be cleared till the CSB is updated and the command stream is switching to the next execution list. If no execution list is pending, the transition of this bit from one to zero guarentees there will be no preemption on the next submission.

Value	Name
0	Invalid **[Default]**
1	Valid

2 — Execlist Queue Full

When [Execlist Write Pointer] and [Current Execlist Pointer] are equal, this bit differentiates between Queue Full and Queue Empty.

Value	Name	Description
0	Execlist Queue Empty **[Default]**	
1	Execlist Queue Full	There is a current and a pending execlist.

1 — Execlist Write Pointer

Format:	ExeclistContentsIndex

Determines which Execlist will be the next submitted to. When a new execlist is submitted, this pointer increments to point to the next execlist slot.

0 — Current Execlist Pointer

Default Value:	1h
Format:	ExeclistContentsIndex

Points at the currently executing Execlist (if there is one). This pointer advances when the first context of new execlist is restored.

Execlist Submit Port Register

EXECLIST_SUBMITPORT - Execlist Submit Port Register	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	02230h-02233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_RCSUNIT
Address:	12230h-12233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VCSUNIT0
Address:	1A230h-1A233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VECSUNIT
Address:	1C230h-1C233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VCSUNIT1
Address:	22230h-22233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_BCSUNIT

SW should submit a new pending execlist to this register. The DWs of the context descriptors must be written in a specific order: Element 1 must be written first and then Element 0. For each Element, DW1 must be written first followed by DW0. Context descriptors for both the elements must be written even if only one context are being submitted. The valid bits of the unused context descriptors should be set to 0.

Order of DW Submission to the Execlist Port
Element 1, High Dword
Element 1, Low Dword
Element 0, High Dword
Element 0, Low Dword

If a execlist of only one element is being submitted, it must be submitted in Element 0. It is UNDEFINED to submit a execlist with the valid bit of Element 0 clear (an "empty" execlist). It is possible that one or all of the contexts submitted in a execlists are "empty"; that is, have head and tail pointers equal to each other indicating no commands to be run. All of the valid bits in the Execlist Element Status Registers for the "about to be submitted" execlist will be cleared when the first DW (DW1 of Element 1) is written to the submit port. Submission of the Element 0 Context Descriptor low Dword with the valid bit set is interpreted as a request to switch (as soon as possible) to the new execlist, i.e., a pre-emption request.

EXECLIST_SUBMITPORT - Execlist Submit Port Register

If a submitted Execlist's Element 0 Context Descriptor LRCA matches the LRCA of the currently executing context, then the newly submitted execlist will become the currently executing execlist without any context switch and without any impact to the executing context except that it will re-sample the tail pointer from the context image. This is done in case more commands have been inserted into its ring buffer between the first execlist submission and the 2nd.

Programming Notes
SW must ensure the contexts submitted to the both the context descriptors in the execlist are different, i.e SW must not submit the same context descriptor to both the elements of the execlist.

DWord	Bit	Description
0	31:0	**Context Descriptor DW** Format: Context Descriptor See "Context Descriptor Format" for format. The element that this DW is submitted as and whether it is the high DW or the low DW is determined by order. This register must be written to 4 times in order to submit a execlist.

Execute Condition Code Register

EXCC - Execute Condition Code Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1

Address:	02028h-0202Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_RCSUNIT

Address:	12028h-1202Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VCSUNIT0

Address:	1A028h-1A02Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VECSUNIT

Address:	1C028h-1C02Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VCSUNIT1

Address:	22028h-2202Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_BCSUNIT

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a 1, while instruction is discarded if the condition evaluates to a 0. Once excluded a ring is enabled into arbitration when the selected condition evaluates to a 0.

This register also contains control for the invalidation of indirect state pointers on context restore.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask These bits serves as a write enable for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15	**Reserved** Format: MBZ

EXCC - Execute Condition Code Register

	14	**Context Wait for V-blank on Pipe-C** Source: RenderCS, BlitterCS This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe C Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	13	**Context Wait for V-blank on Pipe-B** Source: RenderCS, BlitterCS This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe B Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	14:12	**Reserved** Source: VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	12	**Context Wait for V-blank on Pipe-A** Source: RenderCS, BlitterCS This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe A Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	11:5	**Reserved** Format: MBZ
	4:0	**User Defined Condition Codes** Source: RenderCS The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).
	4:0	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ

FAULT_TLB_RD_DATA0 Register

FAULT_TLB_RD_DATA0 - FAULT_TLB_RD_DATA0 Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04B10h	
DWord	**Bit**	**Description**
0	31:0	**FAULT_TLB_READ_DATA0 Register** Default Value: 00000000h Access: RO Fault cycle Virtual address [43:12]

FAULT_TLB_RD_DATA1 Register

FAULT_TLB_RD_DATA1 - FAULT_TLB_RD_DATA1 Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04B14h	
DWord	**Bit**	**Description**
0	31:0	**FAULT_TLB_READ_DATA1 Register** Default Value: 00000000h Access: RO Bit[31:5] Reserved Bit[4] Cycle GTT SEL (1-GGTT Cycle, 0-PPGTT Cycle) Bit[3:0] Fault cycle Virtual address [47:44]

Fault Mode Control

FLTMODECTL - Fault Mode Control		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0404Ch	
DWord	**Bit**	**Description**
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Fault Halt Enable Bit** Default Value: 0b Access: R/W When set, it would enable the Fault and Halt behavior for streamable clients. Page walker will no longer use Fault and Stream mode for any client, instead it will downgrade the fault treatment to fault and halt. This behavior is applicable to HDC/Sampler/I$ given they are the only page fault streamable interfaces. This bit is only applicable under advanced context when PFM is selected for Fault and Stream.

Fault Mode Control

FAULT_MODE_CONTROL - Fault Mode Control		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0404Ch	
DWord	**Bit**	**Description**
0	31:1	**Reserved** Format: MBZ
	0	**Fault and Halt Enable** Format: Enable When set, it would enable the Fault and Halt behavior for streamable clients. Page walker will no longer use Fault and Stream mode for any client, instead it will downgrade the fault treatment to Fault and Halt. This behavior is applicable to HDC/Sampler/I$ given they are the only page fault streamable interfaces. *This bit is only applicable under advanced context when PFM is selected for Fault and Stream.*

Fault Switch Out

FAULT_SO - Fault Switch Out		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04590h	
DWord	**Bit**	**Description**
0	31:0	**Fault Switch Out** Default Value: 00000000h Access: R/W

FBC_CFB_BASE

FBC_CFB_BASE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	43200h-43203h
Name:	FBC Compressed Buffer Address
ShortName:	FBC_CFB_BASE
Power:	PG1
Reset:	soft

Restriction

Restriction : The contents of this register must not be changed while compression is enabled.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:12	**CFB Offset Address** This register specifies bits 27:12 of the offset of the Compressed Frame Buffer from the base of stolen memory. **Restriction** Restriction : The buffer must be 4K byte aligned. The offset must be greater than 4K bytes, avoiding the first 4KB of stolen memory.
	11:0	**Reserved** Format: MBZ

FBC_CTL

FBC_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	43208h-4320Bh
Name:	FBC Control
ShortName:	FBC_CTL
Power:	PG1
Reset:	soft

Description

FBC is tied to Plane 1 A.

Programming Notes

Frame Buffer Compression is only supported with surfaces up to 4096 pixels x 4096 lines and plane sizes up to 4096 pixels x 4096 lines.

The FBC compressed vertical limit is 2560 lines, after which the remaining lines will be displayed correctly, but will not be compressed.

Restriction

Restriction : The contents of this register must not be changed, except the enable bit, while compression is enabled. Frame Buffer Compression is only supported with 16bpp and 32bpp 8:8:8 RGB plane source pixel formats. It is not supported with any other format. The 16bpp format requires the compression ratio to be set to 2:1 or 4:1.

Restriction :
Frame Buffer Compression is not supported with interlaced fetch.
With plane 90/270 rotation, all frame buffer modifications will result in full frame invalidation and recompression. FBC should not be enabled with RGB 16bpp plane formats when plane 90/270 rotation is enabled.
Frame Buffer Compression is not supported when the plane width is smaller than 35 pixels.

DWord	Bit	Description
0	31	**Enable FBC** This bit is used to globally enable FBC function at the next Vertical Blank start. FBC should not be enabled when the pipe is disabled.

Value	Name
0b	Disable
1b	Enable

FBC_CTL

Workaround

When FBC is enabled and the plane surface format is in linear, tile Y legacy or tile YF, the display register 4208Ch bit 13 must be set to 1b and bits 12:0 must be programmed with the compressed buffer stride value.
The compressed buffer stride must be calculated using the following equation:
Compressed buffer stride = ceiling [(at least plane width in pixels) / (32 * compression limit factor)] * 8
At least plane width = a value greater than or equal to the width of the plane. Software may choose to use a greater value in order to handle cases where the plane width is changing from frame to frame, especially because 4208C is not double-buffered and can't be changed on the fly while FBC is enabled. Compression limit factor is either 1, 2 or 4 based on the Compression Limit field.

30:29 **Reserved**

Format:	MBZ

28 **CPU Fence Enable**

Value	Name	Description
0b	No CPU Disp Buf	Display Buffer is not in a CPU fence. No modifications are allowed from CPU to the Display Buffer.
1b	CPU Disp Buf	Display Buffer exists in a CPU fence.

27:25 **Reserved**

Format:	MBZ

24:16 **Reserved**

15 **Reserved**

14:11 **Reserved**

Format:	MBZ

10 **Reserved**

9:8 **Reserved**

7:6 **Compression Limit**

This register sets a minimum limit on compression. This determines the maximum size of the compressed frame buffer. Display lines that do not meet the compression limit will not be compressed, so the best compression will be achieved with a 1:1 ratio.

Compression Ratio 1, Pixel Format 16 bpp - Not Supported
Compression Ratio 1, Pixel Format 32 bpp - Supported (CFB=FB)
Compression Ratio 1/2, Pixel Format 16 bpp - Supported (CFB=FB)
Compression Ratio 1/2, Pixel Format 32 bpp - Supported (CFB=1/2 FB)
Compression Ratio 1/4, Pixel Format 16 bpp - Supported (CFB=1/2 FB)
Compression Ratio 1/4, Pixel Format 32 bpp - Supported (CFB=1/4 FB)

FB = Frame Buffer Size
CFB = Compressed Frame Buffer Size

Value	Name	Description

FBC_CTL

00b	1:1	Compressed buffer is the same size as the uncompressed buffer.
01b	2:1	Compressed buffer is one half the size of the uncompressed buffer.
10b	4:1	Compressed buffer is one quarter the size of the uncompressed buffer.
11b	Reserved	Reserved

5:4 **Write Back Watermark**

The compressed data write back engine waits for this number of entries to be ready before writing the data out to memory.

Value	Name	Description
00b	4	4 entries
01b	8	8 entries
10b	16	16 entries
11b	32	32 entries

3:0 **CPU Fence Number**

Value	Name
0000b	Fence 0

Restriction
Restriction : This field must be programmed to 0000b.

FBC_RT_BASE_ADDR_REGISTER

FBC_RT_BASE_ADDR_REGISTER - FBC_RT_BASE_ADDR_REGISTER

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07020h

This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:12	**FBC RT Base Address** Access: R/W Format: PPGraphicsAddress[31:12] 4KB aligned Base Address as mapped in the PPGTT or in the GGTT for the render target. This base address must be the one that is either front buffer or the back-buffer (a flip target). It must be programmed before any draw call binding that render target base address.
	11:2	**Reserved** Access: R/W Format: PBC
	1	**FBC Front Buffer Target** Access: R/W Format: Enable Value / Name / Description: 0h **[Default]** FBC is targeting the Back Buffer for compression. This buffer can be cached in the MLC/LLC, so a GFDT flush is required before FBC can begin compression. 1h FBC is targeting the Font Buffer for compression. This buffer cannot be cached in the MLC/LLC. FBC compression can begin after any RC flush.
	0	**PPGTT Render Target Base Address Valid for FBC** Access: R/W Format: Enable Value / Name / Description: 0h **[Default]** Base address in this register [31:12] is not valid and therefore FBC will not get any modifications from rendering. 1h Base address in this register [31:12] is valid and HW needs to compare the current render target base address with this base address to provide modifications to FBC.

FBC_RT_BASE_ADDR_REGISTER_UPPER

FBC_RT_BASE_ADDR_REGISTER_UPPER - FBC_RT_BASE_ADDR_REGISTER_UPPER

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07024h

This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:16	**Reserved** Access: R/W Format: PBC
	15:0	**FBC RT Base Address High** Access: R/W Format: BaseAddress[47:32] Must be set to modify corresponding data bit. Reads to this field returns zero. Upper 4KB aligned Base Address as mapped in the PPGTT or in the GGTT for the render target. This base address must be the one that is either front buffer or the back-buffer (a flip target). It can be only programmed once per context. **Programming Notes** It must be programmed before any draw call binding that render target base address.

FBC LLC Config Read Control Register

FBC_LLC_READ_CTRL - FBC LLC Config Read Control Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x000000FF
Size (in bits):	32
Address:	09044h

Frame Buffer Cache Readiness Status Poll Config Register. ONly applicable for KBL family

DWord	Bit	Description
0	31	**FBC LLC Config Read Control Register Lock** Access: R/W Lock 0 = Bits of FBC_CTRL Register are R/W. 1 = All bits of FBC_CTRL Register are RO (including this lock bit). Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 does not clear the lock). These bits are not reset on FLR.
	30	**FBC LLC Config Start Value** Access: R/W Lock PCU_CR_LLC_CONFIG Read Cycle Interval in microseconds. 1'b0 - Treat LLC as partially open on reset (boot or C6 exit) (Default). 1'b1 - Treat LLC as fully open on reset (boot or C6 exit). This must not be set unless coordinated with Uncore.
	29:16	**Reserved** Access: RO Reserved.
	15:0	**FBC LLC Config Read Interval** Default Value: 00FFh Access: R/W Lock PCU_CR_LLC_CONFIG Read Cycle Interval in microseconds . 0x0000: Do not read PCU_CR_LLC_CONFIG (use Start Value only). 0x0001-0xFFFF : Read PCU_CR_LLC_CONFIG at the specified interval, until LLC_FULLY_OPEN=1. Default: 0xFF (approx 170us).

Fence Control Register

MFCR - Fence Control Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00070000
Size (in bits):	32
Address:	09070h

Fence Control Register

DWord	Bit	Description
0	31	**Fuse Override Lock** Access: R/W Lock SW Fuse Override Lock Bit
	30:25	**ECORSVD** Access: R/W ECO purposes Reserved
	24:23	**GT VBOX DISABLE FUSE OVERRIDE** Access: R/W Lock S/W GT Vbox Disable Fuse Override Bits
	22	**Reserved**
	21:19	**GT SUBSLICE DISABLE FUSE OVERRIDE** Access: R/W Lock SW GT SubSlice Disable Fuse Override Bits
	18:16	**GT SLICE ENABLE FUSE OVERRIDE** Default Value: 111b Access: R/W Lock SW GT Slice Enable Fuse Override Bits
	15:5	**RSVD** Access: RO
	4	**Reserved**
	3	**Reserved**
	2	**Write/Read Port Block** Access: R/W 0 - Dont Block the R/W port when Query is started. 1 - Block the R/W port until the Memory Fence is completed. This is applicable for only Memory Fence.

MFCR - Fence Control Register

	1	**LLC Query Enable** Access: R/W 0 - Query for 16 Ways. 1 - Query for 32 Ways. No Flexing.
	0	**Fence Controller GFDT Mode** Access: R/W Fence Controller GFDT Mode. 0 - Single bit GFDT mode. 1 - Two bit GFDT mode.

FF Performance

FF_PERF - FF Performance

Register Space:	MMIO: 0/2/0
Source:	RenderCS, PositionCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	06B1Ch
Name:	Render CS FF Performance
ShortName:	RCS_FF_PERF

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:11	**Reserved** Access: r/w Format: PBC
	10:8	**Throttle counter value** Access: r/w Format: Disable Counter value defining how many clocks the interface needs to be slowed down. Value: 0h — Name: **[Default]** — Description: Masked by default.
	7:3	**Reserved** Access: r/w Format: PBC
	2	**Enable throttling for SF-WM interface** Access: r/w Format: Disable Value: 0h — Name: Disable — Description: No throttling Value: 1h — Name: Enable — Description: Enable throttling in all SF-WM interfaces

FF_PERF - FF Performance

1 Enable throttling for SF-SBE interface

Access:	r/w
Format:	Disable

Value	Name	Description
0h	Disable	No throttling
1h	Enable	Enable throttling in all SF-SBE interfaces

0 Enable throttling for CL-SF interface

Access:	r/w
Format:	Disable

Value	Name	Description
0h	Disable	No throttling
1h	Enable	Enable throttling in all CL-SF interfaces

Restriction
Restriction : This bit must not be set. SW may choose to use SF-SBE throttle interface(bit 1) to achive the same effect.

First Buffer Size and Start

FBSS - First Buffer Size and Start

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B420h

LPFCREG02 - First Buffer Size and Start

DWord	Bit	Description
0	31:16	**First Virtual Buffer Base** Access: R/W First Virtual Buffer Base: Programmed by driver to allocate a memory space for performance data storage. The buffer size should be aligned to the size of the memory allocated so it naturally aligns to the base (i.e. for 128KB bit[16]=0, 256KB bit[17:16]=0, 512KB bit[18:16]=0). Signal - lpconf_lpfc_virtual_base0 [31:16].
	15:12	**First Buffer Size** Access: R/W First Buffer Size: Determines the allowed buffer size for performance data storage. 0000b: 64KB. 0001b: 128KB. 0010b: 256KB. 0011b: 512KB. ... 1111b: 2GB. Signal - lpconf_lpfc_buffer_size0 [3:0].
	11:4	**Reserved** Access: RO Reserved.
	2	**Frame count and Draw call enable** Access: R/W Enables the replacement of a specific L3 performance counter value in the reported data with a 16-bit tag created from the concatenation of the "Frame Count" and "Draw Call Number" programmable bitfields in the "Frame Count and Draw Call Number" register. The exact counter replaced is dependent on the programmed value of the "Counter Enabling Selection" bitfield. The replaced counter is always the last one, except in the case only a single performance counter is enabled for reporting (in which no replacement occurs): CNTRENSEL Value / Replaced Event Counter 00 / No Replacement 01 / Counter 1 10 / Counter 3

FBSS - First Buffer Size and Start

		11 Counter 7
	1	**Reserved**
	0	**Master Counter Enable** Access: R/W Master Counter Enable: This is the global enable for performance tracking. Once set, it kicks off all performance tracking mechanism. Signal - lpconf_lpfc_master_cnt_en. This bit is used by all slices.

Flexible EU Event Control 0

EU_PERF_CNT_CTL0 - Flexible EU Event Control 0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E458h

This register configures flexible EU event 0/1. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 1** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 1. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 1** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 1. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 1** Format: U4 This field controls which increment event provides the basis for flexible EU event 1.
	11:8	**Fine Event Filter Select EU event 0** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 0. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 0** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 0. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 0** Format: U4 This field controls which increment event provides the basis for flexible EU event 0.

Flexible EU Event Control 1

EU_PERF_CNT_CTL1 - Flexible EU Event Control 1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E558h

This register configures flexible EU event 2/3. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 3** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 3. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 3** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 3. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 3** Format: U4 This field controls which increment event provides the basis for flexible EU event 3.
	11:8	**Fine Event Filter Select EU event 2** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 2. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 2** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 2. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 2** Format: U4 This field controls which increment event provides the basis for flexible EU event 2.

Flexible EU Event Control 2

EU_PERF_CNT_CTL2 - Flexible EU Event Control 2	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E658h

This register configures flexible EU event 4/5. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 5** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 5. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 5** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 5. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 5** Format: U4 This field controls which increment event provides the basis for flexible EU event 5.
	11:8	**Fine Event Filter Select EU event 4** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 4. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 4** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 4. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 4** Format: U4 This field controls which increment event provides the basis for flexible EU event 4.

Flexible EU Event Control 3

EU_PERF_CNT_CTL3 - Flexible EU Event Control 3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E758h

This register configures flexible EU event 6/7. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 7** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 7. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 7** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 7. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 7** Format: U4 This field controls which increment event provides the basis for flexible EU event 7.
	11:8	**Fine Event Filter Select EU event 6** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 6. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 6** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 6. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 6** Format: U4 This field controls which increment event provides the basis for flexible EU event 6.

Flexible EU Event Control 4

EU_PERF_CNT_CTL4 - Flexible EU Event Control 4	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E45Ch

This register configures flexible EU event 8/9. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 9** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 9. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 9** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 9. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 9** Format: U4 This field controls which increment event provides the basis for flexible EU event 9.
	11:8	**Fine Event Filter Select EU event 8** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 8. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 8** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 8. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 8** Format: U4 This field controls which increment event provides the basis for flexible EU event 8.

Flexible EU Event Control 5

EU_PERF_CNT_CTL5 - Flexible EU Event Control 5

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E55Ch

This register configures flexible EU event 10/11. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 11** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 11. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 11** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 11. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 11** Format: U4 This field controls which increment event provides the basis for flexible EU event 11.
	11:8	**Fine Event Filter Select EU event 10** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 10. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 10** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 10. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 10** Format: U4 This field controls which increment event provides the basis for flexible EU event 10.

Flexible EU Event Control 6

EU_PERF_CNT_CTL6 - Flexible EU Event Control 6	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E65Ch

This register configures flexible EU event 12/13. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:20	**Fine Event Filter Select EU event 13** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 13. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 13** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 13. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 13** Format: U4 This field controls which increment event provides the basis for flexible EU event 13.
	11:8	**Fine Event Filter Select EU event 12** Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 12. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 12** Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 12. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 12** Format: U4 This field controls which increment event provides the basis for flexible EU event 12.

FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00002094
Access:	R/W
Size (in bits):	32
Address:	024D0h-024D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_RCSUNIT
Address:	024D4h-024D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_RCSUNIT
Address:	024D8h-024DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_RCSUNIT
Address:	024DCh-024DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_RCSUNIT
Address:	024E0h-024E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_RCSUNIT
Address:	024E4h-024E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_RCSUNIT
Address:	024E8h-024EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_RCSUNIT
Address:	024ECh-024EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_RCSUNIT
Address:	024F0h-024F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_RCSUNIT
Address:	024F4h-024F7h

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_RCSUNIT
Address:	024F8h-024FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_RCSUNIT
Address:	024FCh-024FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_RCSUNIT
Address:	124D0h-124D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VCSUNIT0
Address:	124D4h-124D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VCSUNIT0
Address:	124D8h-124DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VCSUNIT0
Address:	124DCh-124DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VCSUNIT0
Address:	124E0h-124E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VCSUNIT0
Address:	124E4h-124E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VCSUNIT0
Address:	124E8h-124EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VCSUNIT0
Address:	124ECh-124EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VCSUNIT0
Address:	124F0h-124F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_VCSUNIT0

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Address:	124F4h-124F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VCSUNIT0
Address:	124F8h-124FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VCSUNIT0
Address:	124FCh-124FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VCSUNIT0
Address:	1A4D0h-1A4D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VECSUNIT
Address:	1A4D4h-1A4D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VECSUNIT
Address:	1A4D8h-1A4DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VECSUNIT
Address:	1A4DCh-1A4DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VECSUNIT
Address:	1A4E0h-1A4E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VECSUNIT
Address:	1A4E4h-1A4E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VECSUNIT
Address:	1A4E8h-1A4EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VECSUNIT
Address:	1A4ECh-1A4EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VECSUNIT
Address:	1A4F0h-1A4F3h
Name:	FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
ShortName:	FORCE_TO_NONPRIV_8_VECSUNIT
Address:	1A4F4h-1A4F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VECSUNIT
Address:	1A4F8h-1A4FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VECSUNIT
Address:	1A4FCh-1A4FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VECSUNIT
Address:	1C4D0h-1C4D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VCSUNIT1
Address:	1C4D4h-1C4D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VCSUNIT1
Address:	1C4D8h-1C4DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VCSUNIT1
Address:	1C4DCh-1C4DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VCSUNIT1
Address:	1C4E0h-1C4E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VCSUNIT1
Address:	1C4E4h-1C4E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VCSUNIT1
Address:	1C4E8h-1C4EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VCSUNIT1
Address:	1C4ECh-1C4EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VCSUNIT1

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Address:	1C4F0h-1C4F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_VCSUNIT1
Address:	1C4F4h-1C4F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VCSUNIT1
Address:	1C4F8h-1C4FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VCSUNIT1
Address:	1C4FCh-1C4FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VCSUNIT1
Address:	224D0h-224D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_BCSUNIT
Address:	224D4h-224D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_BCSUNIT
Address:	224D8h-224DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_BCSUNIT
Address:	224DCh-224DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_BCSUNIT
Address:	224E0h-224E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_BCSUNIT
Address:	224E4h-224E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_BCSUNIT
Address:	224E8h-224EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_BCSUNIT
Address:	224ECh-224EFh
Name:	FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

ShortName:	FORCE_TO_NONPRIV_7_BCSUNIT
Address:	224F0h-224F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_BCSUNIT
Address:	224F4h-224F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_BCSUNIT
Address:	224F8h-224FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_BCSUNIT
Address:	224FCh-224FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_BCSUNIT

These registers are privilege registers and are not allowed to be written from non-privilege batch buffer. These are global registers and power context save/restored.

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30:26	**Reserved** Format: MBZ
	25:2	**Non Privilege Register Address** Format: MmioAddress[25:2] This field contains the MMIO offset of a register. MMIO offset programmed in this field will be treated as a non-privilege register by render command streamer while processing register writes from a non-privilege batch buffer. This register provides programmability is to extend the non-privilege register table mentioned in MI_BATCH_BUFFER_START command in render command streamer. Value: 825h — Name: **[Default]**
	1:0	**Reserved** Format: MBZ

Frame Buffer Cache Definition Register

FBCDR - Frame Buffer Cache Definition Register

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04068h

DWord	Bit	Description
0	31:25	**Reserved** Default Value: 0000000b Access: R/W
	24:23	**Arbitration of LLCWBInv v/s Memory Writes** Default Value: 00b Access: R/W A round robin is enabled to inject LLCWbInv cycles to memory write stream. The conflicts are handled with programming the round robin mechanism. 00b: 1 memory write and 1 LLCWbInv. 01b: 2 memory writes and 1 LLCWbInv 10b: 4 memory writes and 1 LLCWbInv 11b: 8 memory writes and 1 LLCWbInv
	22:18	**Inside CBO** Default Value: 00000b Access: R/W CBO ID is added in LLC to separate the banks of LLC in a given slice. GFX driver will set the corresponding bits to enable the toggling of CBOID during flushing [22]: Enable h/w to toggle of CBOID[4] during flush. [21]: Enable h/w to toggle of CBOID[3] during flush. [20]: Enable h/w to toggle of CBOID[2] during flush. [19]: Enable h/w to toggle of CBOID[1] during flush. [18]: Enable h/w to toggle of CBOID[0] during flush. Setting the corresponding bit would mean h/w has to permute the corresponding bit for CBOID during flush, else h/w will keep the corresponding bit as "0" Note: For SKL 2-core system, only bit[18] needs to be set. For SKL 4-core system, bits [19:18] needs to be set.
	17:16	**Inside CBOID Enable** Default Value: 00b Access: R/W Inside CBO ID is added in LLC to separate the banks of LLC in a given slice. GFX driver will set the corresponding bits to enable the toggling of Inside CBOID during flushing:

FBCDR - Frame Buffer Cache Definition Register

		[17]: Enable h/w to toggle of InsideCBOID[1] during flush. [16]: Enable h/w to toggle of InsideCBOID[0] during flush. Setting the corresponding bit would mean h/w has to permute the corresponding bit for InsideCBOID during flush, else h/w will keep the corresponding bit as "0". Note: For SKL client this field needs to be programmed as "01"
	15:0	**Frame Buffer Caching enabled ways** Default Value: 0000h Access: R/W The Last Level cache is a 16 way structure where some of the ways are armed to allocate Frame Buffer Data. This determination is done via assigning a QOS value for Frame Buffer and QOS assignments for ways. Each bit corresponds to a way in this definition. Bit[15]: Way#15 is enabled to allocate Frame Buffer Data. Bit[14]: Way#14 is enabled to allocate Frame Buffer Data. Bit[13]: Way#13 is enabled to allocate Frame Buffer Data. Bit[12]: Way#12 is enabled to allocate Frame Buffer Data. Bit[11]: Way#11 is enabled to allocate Frame Buffer Data. Bit[10]: Way#10 is enabled to allocate Frame Buffer Data. Bit[9]: Way#9 is enabled to allocate Frame Buffer Data. Bit[8]: Way#8 is enabled to allocate Frame Buffer Data. Bit[7]: Way#7 is enabled to allocate Frame Buffer Data. Bit[6]: Way#6 is enabled to allocate Frame Buffer Data. Bit[5]: Way#5 is enabled to allocate Frame Buffer Data. Bit[4]: Way#4 is enabled to allocate Frame Buffer Data. Bit[3]: Way#3 is enabled to allocate Frame Buffer Data. Bit[2]: Way#2 is enabled to allocate Frame Buffer Data. Bit[1]: Way#1 is enabled to allocate Frame Buffer Data. Bit[0]: Way#0 is enabled to allocate Frame Buffer Data.

Frame count and Draw call number

FCDCN - Frame count and Draw call number

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B430h

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:8	**Frame Number** Access: R/W Frame number is the first of two reporting tags that software (i.e. driver) may populate in order to provide reference points during L3 performance reporting modes. Should the "Frame Count and Draw Call Enable" bit (FCDCE) in the "First Buffer Size and Start" register be set, LPFC will selectively replace one of the reporting events with this programmable tag (in addition to the "Draw Call Number" field below). Software may use this to provide reference points for L3 performance counts when parsing the resulting data stream to align reported counts to higher-level operations. The original incarnation called for software to increment this value with each frame, however, the field is generic and may be used for any tagging purpose.
	7:0	**Draw call number** Access: R/W The draw call number is the second programmable reporting tag provided by LPFC. With this second programmable tag, a more granular sampling boundary may be created by software, or it may be used to provide an alternative reference point for tracking L3 performance. The original incarnation called for software to increment this value with every draw call, but the field is generic and may be used for any similar purpose.

FUSE_STATUS

FUSE_STATUS

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	42000h-42003h
Name:	Fuse Status
ShortName:	FUSE_STATUS
Power:	PG0
Reset:	global

This register is on the ungated clock and the chip reset, not the FLR reset.

DWord	Bit	Description
0	31	**Fuse Download Status** Access: RO This field indicates the status of fuse and strap download to the Display Engine. After fuse and strap download, fuses will be distributed within the Display Engine. 0b – Not Done 1b – Done
	30:28	**Reserved**
	27	**Fuse PG0 Distribution Status** Access: RO This field indicates the status of fuse distribution to power well #0. 0b – Not Done 1b – Done
	26	**Fuse PG1 Distribution Status** Access: RO This field indicates the status of fuse distribution to power well #1. 0b – Not Done 1b – Done
	25	**Fuse PG2 Distribution Status** Access: RO This field indicates the status of fuse distribution to power well #2.

FUSE_STATUS

		Value	Name
		0b	Not Done
		1b	Done
	24:0	**Reserved**	

GAB Arbitration Programmable

GAB_AP - GAB Arbitration Programmable		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	040F0h	
DWord	**Bit**	**Description**
0	31:0	**Reserved**

GAB LRA 0

GAB_LRA_0 - GAB LRA 0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x1F100F00
Size (in bits):	32
Address:	04A70h

DWord	Bit	Description
0	31:29	**Reserved** Default Value: 000b Access: RO
	28:24	**GAB LRA1 Max** Default Value: 11111b Access: R/W Maximum value of programmable LRA1.
	23:21	**Reserved** Default Value: 000b Access: RO
	20:16	**GAB LRA1 Min** Default Value: 10000b Access: R/W Minimum value of programmable LRA1.
	15:13	**Reserved** Default Value: 000b Access: RO
	12:8	**GAB LRA0 Max** Default Value: 01111b Access: R/W Maximum value of programmable LRA0.
	7:5	**Reserved** Default Value: 000b Access: RO

GAB_LRA_0 - GAB LRA 0

	4:0	**GABLRA0 Min** Default Value: 00000b Access: R/W Minimum value of programmable LRA0.

GAB LRA 1

GAB_LRA_1 - GAB LRA 1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000001
Size (in bits):	32
Address:	04A74h

DWord	Bit	Description
0	31:4	**Reserved** Default Value: 0000000h Access: RO
	3:2	**BLB** Default Value: 00b Access: R/W Which LRA should BLB use.
	1:0	**BCS** Default Value: 01b Access: R/W Which LRA should BCS use.

GAB unit Control Register

GAB_CTL_REG - GAB unit Control Register

Register Space:	MMIO: 0/2/0
Source:	BlitterCS
Default Value:	0x000000BF
Access:	R/W
Size (in bits):	32
Address:	24000h

DefaultValue=FF0000BFh Trusted Type = 1

DWord	Bit	Description
0	31:9	**Reserved**
	8	**Continue after Page Fault** Value / Name / Description: 1 / GAB Set / Ipon receiving a page fault when requesting an address translation, GAB will set address bit 39 to 1 and continue. 0 / GAB Hang / GAB will hang on a page fault. Default = b0.
	7:6	**PPGTT BCS TLB LRA MIN** Default Value: 10b TLB Depth Partitioning Register In PP GTT Mode.
	5:4	**GAB write request priority signal value used in GAC arbitration** Default Value: 11b
	3:2	**GAB read only request priority signal value used in GAC arbitration** Default Value: 11b
	1:0	**GAB read request priority signal value used in GAC arbitration** Default Value: 11b

GAC_GAM Arbitration Counters Register 0

ARB_GAC_GAM_REQCNTS0 - GAC_GAM Arbitration Counters Register 0		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043A8h	
DWord	**Bit**	**Description**
0	31:22	**Reserved**
	21:16	**Number of GAC WR requests to be accumulated before applying the arbitration**
	15:14	**Reserved**
	13:8	**Number of GAC R requests to be accumulated before applying the arbitration**
	7:6	**Reserved**
	5:0	**Number of GAC RO requests to be accumulated before applying the arbitration**

GAC_GAM Arbitration Counters Register 1

ARB_GAC_GAM_REQCNTS1 - GAC_GAM Arbitration Counters Register 1		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043ACh	
DWord	**Bit**	**Description**
0	31:22	**Reserved**
	21:16	**Number of GAC WR requests to be accumulated before applying the arbitration**
	15:14	**Reserved**
	13:8	**Number of GAC R requests to be accumulated before applying the arbitration**
	7:6	**Reserved**
	5:0	**Number of GAC RO requests to be accumulated before applying the arbitration**

GAC_GAM R Arbitration Register 0

ARB_R_GAC_GAM0 - GAC_GAM R Arbitration Register 0		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043E0h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 0**
	11:9	**Goto field for entry 0 when request vector is 11b**
	8:6	**Goto field for entry 0 when request vector is 10b**
	5:3	**Goto field for entry 0 when request vector is 01b**
	2:0	**Goto field for entry 0 when request vector is 00b**

GAC_GAM R Arbitration Register 1

ARB_R_GAC_GAM1 - GAC_GAM R Arbitration Register 1		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043E4h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM R Arbitration Register 2

ARB_R_GAC_GAM2 - GAC_GAM R Arbitration Register 2		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043E8h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM R Arbitration Register 3

ARB_R_GAC_GAM3 - GAC_GAM R Arbitration Register 3		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043ECh	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAC_GAM RO Arbitration Register 0

ARB_RO_GAC_GAM0 - GAC_GAM RO Arbitration Register 0		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043D0h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 01**
	11:9	**Goto field for entry 01 when request vector is 11b**
	8:6	**Goto field for entry 01 when request vector is 10b**
	5:3	**Goto field for entry 01 when request vector is 01b**
	2:0	**Goto field for entry 01 when request vector is 00b**

GAC_GAM RO Arbitration Register 1

ARB_RO_GAC_GAM1 - GAC_GAM RO Arbitration Register 1		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043D4h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM RO Arbitration Register 2

ARB_RO_GAC_GAM2 - GAC_GAM RO Arbitration Register 2		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043D8h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM RO Arbitration Register 3

ARB_RO_GAC_GAM3 - GAC_GAM RO Arbitration Register 3		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043DCh	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAC_GAM WR Arbitration Register 0

ARB_WR_GAC_GAM0 - GAC_GAM WR Arbitration Register 0	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043F0h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 0**
	11:9	**Goto field for entry 0 when request vector is 11b**
	8:6	**Goto field for entry 0 when request vector is 10b**
	5:3	**Goto field for entry 0 when request vector is 01b**
	2:0	**Goto field for entry 0 when request vector is 00b**

GAC_GAM WR Arbitration Register 1

ARB_WR_GAC_GAM1 - GAC_GAM WR Arbitration Register 1		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043F4h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM WR Arbitration Register 2

ARB_WR_GAC_GAM2 - GAC_GAM WR Arbitration Register 2		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043F8h	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM WR Arbitration Register 3

ARB_WR_GAC_GAM3 - GAC_GAM WR Arbitration Register 3		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Trusted Type:	1	
Address:	043FCh	
DWord	**Bit**	**Description**
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAFS MAX URB READ EVENT

GAFS_MAX_URBRD - GAFS MAX URB READ EVENT		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	00DB4h	
This register mirrors an accumulating count for Unslice FF control It is enabled by configuration bits in GPMunit and SPMunits. Note that count is never cleared and delta should be calculated by sampling the initial register value at the start and subtracting that value off from the final sample value.		
DWord	**Bit**	**Description**
0	31:0	**Unslice FF Event Count** Access: RO

GAM and SA Communication Register

GAMSACOMREG - GAM and SA Communication Register

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	042A0h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**GAM and SA Communication Register 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**GAM and SA Communication Register 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**GAM and SA Communication Register 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**GAM and SA Communication Register 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GAMSACOMREG - GAM and SA Communication Register

	11	**GAM and SA Communication Register 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	10	**GAM and SA Communication Register 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**GAM and SA Communication Register 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**GAM and SA Communication Register 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**GAM and SA Communication Register 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**GAM and SA Communication Register 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GAMSACOMREG - GAM and SA Communication Register

	5	**GAM and SA Communication Register 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**GAM and SA Communication Register 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**GAM and SA Communication Register 3** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	2	**GAM and SA Communication Register 2** Default Value: 0b Access: R/W Bit2 - Root Table Address Update Request. This bit is self clear.
	1	**GAM and SA Communication Register 1** Default Value: 0b Access: R/W Bit1 - Queued Descriptor Request. This bit is self clear.
	0	**GAM and SA Communication Register 0** Default Value: 0b Access: R/W Bit0 - Context Cache Invalidator Request. This bit is self clear.

GAM Context Save

GAM_CTX - GAM Context Save

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	04FFCh

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15:2	**Reserved** Default Value: 0000h Access: RO Reserved Bits for future use
	1	**Context Save Start - Chunk2** Default Value: 0b Access: R/W Context Save start for chunk2 Bit[1] Context Save Request start 1'b0 : Context save is not being requested 1'b1 : Context save is being requested When a 1 is written to this bit , with the corresponding mask bit set, it will kick off a context save. Once the save is complete the bit will be cleared. GAMunit on receiving this message sends the Chunk-2 (Cachelines following Chunk1) of context image to CS
	0	**Context Save Start - Chunk1** Default Value: 0b Access: R/W Context Save start for chunk1 Bit[1] Context Save Request start 1'b0 : Context save is not being requested 1'b1 : Context save is being requested When a 1 is written to this bit , with the corresponding mask bit set, it will kick off a context save. Once the save is complete the bit will be cleared. GAMunit on receiving this message sends the Chunk-1 (first 8 Cachelines of its context image) of context image to CS

GAM Context Save Register

GAM_CTX - GAM Context Save Register		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Address:	04FFCh	
This register is used to send messages to enable context saving. This register may not be written from CPU. Bits [0] and [1] getting set are mutually exclusive.		
DWord	**Bit**	**Description**
0	31:16	**Masks** Format: Mask[15:0] A 1 in a bit in this field allows the modification of the corresponding bit in bits 15:0.
	15:2	**Reserved** Format: MBZ
	1	**Context Save Start - Chunk2** Format: Enable When a 1 is written to this bit , with the corresponding mask bit set, it will kick off a context save. Once the save is complete the bit will be cleared. GAMunit on receiving this message sends the Chunk-2 (Cachelines following Chunk1) of context image to CS.
	0	**Context Save Start - Chunk1** Format: Enable When a 1 is written to this bit with the corresponding mask bit set, it will kick off a context save. Once the save is complete the bit will be cleared. GAMunit on receiving this message sends the Chunk-1 (first 8 Cachelines of its context image) of context image to CS.

Gam Fub Done1 Lookup Register

DONE1_REG - Gam Fub Done1 Lookup Register		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0407Ch	
DWord	**Bit**	**Description**
0	31:0	**Gam Fub Done1 Lookup Reg** Default Value: 00000000h Access: RO GAM Done1 signals.

Gam Fub Done Lookup Register

DONE_REG - Gam Fub Done Lookup Register		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	040B0h	
DWord	**Bit**	**Description**
0	31:0	**Gam Fub Done Lookup Reg** Default Value: 00000000h Access: RO 31 CVS Credit Fifo is empty. 30 CVS TLB does not have any cycles. 29 Z Credit fifo is empty. 28 ZTLB does not have any cycles. 27 RCC Credit Fifo is empty. 26 RCC TLB does not have any cycles. 25 L3 Credit fifo is empty. 24 L3 TLB does not have any cycles. 23 VLF Credit fifo is empty. 22 VLF TLB does not have any cycles. 21 CASC Credit fifo empty. 20 CASC TLB does not have any cycles. 19 Miss Fub Done. 18 Read Stream Done. 17 Read Steam Fifo is empty. 16 Recycle Fifo in rstrm is empty. 15 TLB Pend Done. 14 TLB Pend PQ Array is done. 13 TLB pend PB Array is done. 12 Read route fub is done. 11 Gafm Data fifo is empty. 10 GAP data fifo is empty. 9 GAC data fifo is empty. 8 Wrdp is done with all the cycles. 7 Wrdp RID fifo is empty. 6 No hold from midarb to RTSTRM. 5 No hold from TLBPEND to MIDARB. 3 Tied to "1" - to be defined. 2 Fence FSM are IDLE. 1 Non PD Load Done. 0 Tied to "1" - to be defined.

GAMMA_MODE

GAMMA_MODE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	Double Buffered
Size (in bits):	32
Double Buffer Update Point:	Start of vertical blank
Address:	4A480h-4A483h
Name:	Pipe Gamma Mode
ShortName:	GAMMA_MODE_A
Power:	PG1
Reset:	soft
Address:	4AC80h-4AC83h
Name:	Pipe Gamma Mode
ShortName:	GAMMA_MODE_B
Power:	PG2
Reset:	soft
Address:	4B480h-4B483h
Name:	Pipe Gamma Mode
ShortName:	GAMMA_MODE_C
Power:	PG2
Reset:	soft

DWord	Bit	Description	
0	31:16	**Reserved**	
		Format:	MBZ
	15	**Reserved**	
	14:2	**Reserved**	
		Format:	MBZ

GAMMA_MODE

	1:0	**Gamma Mode** This field selects which mode the pipe palette/gamma correction logic works in. Other gamma units, such as in the planes, are unaffected by this bit.

Value	Name	Description
00b	8 bit	8-bit Legacy Palette Mode
01b	10 bit	10-bit Precision Palette Mode
10b	12 bit	12-bit Interpolated Gamma Mode
11b	Split	Split Gamma Mode (separate pipe gamma functions before and after pipe CSC)

GAM Put Delay

GAM_PUT_DLY - GAM Put Delay		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0401Ch	
Number of clocks to wait between puts		
DWord	**Bit**	**Description**
0	31:0	**GAM PUT DELAY** Default Value: 00000000h Access: R/W

GAMT_DONE Register

GAMT_DONE - GAMT_DONE Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	04AC0h

DWord	Bit	Description
0	31:0	**GAMT_DONE Register** Default Value: 00000000h Access: RO 31: vebxtlb_all_done_f 30: cvstlb_all_done_f 29: ztlb_all_done_f 28: l3tlb_all_done_f 27: rcctlb_all_done_f 26: mfxtlb_all_done_f 25: vlftlb_all_done_f 24: bwgtlb_all_done_f 23: gamwrrb_all_done_f 22: mfxsl1tlb_all_done_f 21: vlfsl1tlb_all_done_f 20: bwgtlb_fifo_empty 19: l3tlb_fifo_empty 18: ztlb_fifo_empty 17: rcctlb_fifo_empty 16: cvstlb_fifo_empty 15: vebxtlb_fifo_empty 14: mfxtlb_fifo_empty 13: mfxsl1tlb_fifo_empty 12: vlfsl1tlb_fifo_empty 11: vlftlb_fifo_empty 10: wrdp_gafm_fifo_empty 9: wrdp_gap_fifo_empty 8: wrdp_gacfg_fifo_empty 7: wrdp_cs_fifo_empty 6: wrdp_vecs_fifo_empty 5: wrdp_oacs_fifo_empty 4: wrdp_gacv_fifo_empty 3: Tied to 1 2: Tied to 1 1: Tied to 1 0: Tied to 1

GAMT_ECO_REG_RO_IA

GAMT_ECO_REG_RO_IA - GAMT_ECO_REG_RO_IA	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	04AB4h

DWord	Bit	Description
0	31:0	**GAMTECO_REG_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on DEV reset.

GAMT_ECO_REG_RW_IA

GAMT_ECO_REG_RW_IA - GAMT_ECO_REG_RW_IA

Register Space:	MMIO: 0/2/0
Source:	BSpec
	0x0000AB1B
Size (in bits):	32
Address:	04AB0h

Programmable Request Count - VEBX and BLT

DWord	Bit	Description
0	31	**TLBINV_PRESENT_DIS** Default Value: 0b Access: R/W Bit[31] = 0 : Default TLB invalidation of valid and present entries only Bit[31] = 1 : Default TLB invalidation of valid entries (pre SKL C0 behaviour)
	30	**BYPASS_HDC_INV** Default Value: 0b Access: R/W Bit[30] = 0 : Default wait for all the HDC acks before sending the ack to GAMW Bit[30] = 1 : GAMT does not wait for all ACK in case of reset in progress; kill acks cycles coming during the window
	29:0	**GAMTECO_REG_RW_IA** Default Value: 00000000000000101010110001101 1b Access: R/W Bit[29] = 0 : Enables fixes in GACB unit related to TSV usage of flush_start, fifo_empty signals Bit[29] = 1 : Disables fixes in GACB unit related to TSV usage of flush_start, fifo_empty signals Bit[28] = 0 : Enables a fix in GACBH unit related to flush signals Bit[28] = 1 : Disables a fix in GACBH unit related to flush signals Bit[27:26] = Control bits to select address hash modes in GACBH unit 00 : Hash bit = addr_bit[6] 01 : Hash bit = xor (addr_bits[9], addr_bits[6]) 10 : Hash bit = xor (addr_bits[11 : 6]) 11 : Hash bit = xor (addr_bits[20 : 6]) Bit[25] = Reserved Bit[24] = 0 : Enables a fix in GACBH unit related to flush signals Bit[24] = 1 : Disables a fix in GACBH unit related to flush signals Bit[23] = Reserved Bit[22:19] = Reserved Bit[18] = Enable support for In-Place decompresssion feature Bit[17:16] = Reserved. Bit[15:8] = Number of max outstanding cycles (Misses and Hits not present) that can be allowed to potentially fault = 171.

GAMT_ECO_REG_RW_IA - GAMT_ECO_REG_RW_IA		
		Bit[7:6] = Reserved. Bit[5:0] = Number of max outstanding misses that can be allowed to potentially fault = 27.

GAMT Arbiter Mode Control

GAMTARBMODE - GAMT Arbiter Mode Control

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000 [KBL]
Size (in bits):	32
Address:	04A08h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**GAMT Arbiter Mode Control 15** Default Value: 0b Access: R/W For Future Use
	14	**GAMT Arbiter Mode Control 14** Default Value: 0b Access: R/W 0 - Cache the TLB even if there is a FAULT in GAMW read return. 1 - Don't Cache the TLB if there is a fault in GAMW return.
	13	**GAMT Arbiter Mode Control 13** Default Value: 0b Access: R/W 0 - VEBXTLB clock gate enabled. 1 - VEBXTLB clock gate disabled.
	12	**GAMT Arbiter Mode Control 12** Default Value: 0b Access: R/W 0 - MFXSL1TLB clock gate enabled. 1 - MFXSL1TLB clock gate disabled.
	11	**GAMT Arbiter Mode Control 11** Default Value: 0b Access: R/W 0 - VLFSL1TLB clock gate enabled. 1 - VLFSL1TLB clock gate disabled.

GAMTARBMODE - GAMT Arbiter Mode Control

	10	**GAMT Arbiter Mode Control 10** Default Value: 0b Access: R/W 0 - gamwrrb clock gate enabled. 1 - gamwrrb clock gate disabled.
	9	**GAMT Arbiter Mode Control 9** Default Value: 0b Access: R/W 0 - BWGTLB clock gate enabled. 1 - BWGTLB clock gate disabled.
	8	**GAMT Arbiter Mode Control 8** Default Value: 0b Access: R/W 0 - VLFTLB clock gate enabled. 1 - VLFTLB clock gate disabled.
	7	**GAMT Arbiter Mode Control 7** Default Value: 0b Access: R/W 0 - MFXTLB clock gate enabled. 1 - MFXTLB clock gate disabled.
	6	**GAMT Arbiter Mode Control 6** Default Value: 0b Access: R/W 0 - RCCTLB clock gate enabled. 1 - RCCTLB clock gate disabled.
	5	**GAMT Arbiter Mode Control 5** Default Value: 0b Access: R/W 0 - L3TLB clock gate enabled. 1 - L3TLB clock gate disabled. To update bit 5, a value of 0x00200020 needs to be written.

GAMTARBMODE - GAMT Arbiter Mode Control

	4	**GAMT Arbiter Mode Control 4** Default Value: 0b Access: R/W 0 - ZTLB clock gate enabled. 1 - ZTLB clock gate disabled.
	3	**GAMT Arbiter Mode Control 3** Default Value: 0b Access: R/W 0 - CVS clock gate enabled. 1 - CVS clock gate disabled.
	2	**GAMT Arbiter Mode Control 2** Default Value: 0b Access: R/W 0 - No reg_hdc_inval_ack_force - take the value from client. 1 - reg_hdc_inval_ack_force - force value to 1 - disregard client value.
	1	**GAMT Arbiter Mode Control 1** Default Value: 0b Access: R/W Bit [1]: Address Swizzling for Tiled Surfaces. This register location is updated via GFX Driver prior to enabling DRAM accesses. Driver needs to obtain the need for memory address swizzling via DRAM configuration registers and set the following bits (in Display Engine and Render/Media access streams). 0: No address Swizzling. 1: Address bit[1] needs to be swizzled for tiled surfaces.
	0	**GAMT Arbiter Mode Control 0** Default Value: 0b Access: R/W Bit[0]: GAM to Bypass GTT Translation. GAM to Bypass GTT Translation and pass logical addresses through with 0's padded on the MSBs to form the Physical Address.

GAM Virtualization Enable Register

GAM_VFX_EN - GAM Virtualization Enable Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	041FCh

This register enables virtualization for GAM

DWord	Bit	Description
0	31:1	**Extra Bits** Default Value: 00000000h Access: R/W Reserved for future use
	0	**Virtualization Enable** Default Value: 0b Access: R/W 1'b0 -- Virtualization is not enabled 1'b1 -- Virtualization is enabled

GAMW_ECO_BUS_RO_IA

GAMW_ECO_BUS_RO_IA - GAMW_ECO_BUS_RO_IA		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0408Ch	
DWord	**Bit**	**Description**
0	31:0	**GAMWECO_BUS_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on BUS reset.

GAMW_ECO_BUS_RW_IA

GAMW_ECO_BUS_RW_IA - GAMW_ECO_BUS_RW_IA		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04084h	
DWord	**Bit**	**Description**
0	31:0	**GAMWECO_BUS_RW_IA** Default Value: 00000000h Access: R/W This register is for ECO usage. RW register with IA Access Type on BUS reset.

GAMW_ECO_DEV_RO_IA

GAMW_ECO_DEV_RO_IA - GAMW_ECO_DEV_RO_IA		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04088h	
DWord	**Bit**	**Description**
0	31:0	**GAMWECO_DEV_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on DEV reset.

GAMW_ECO_DEV_RW_IA

GAMW_ECO_DEV_RW_IA - GAMW_ECO_DEV_RW_IA		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04080h	
DWord	**Bit**	**Description**
0	31:0	**GAMWECO_DEV_RW_IA** Default Value: 00000000h Access: R/W This register is for ECO usage. RW register with IA Access Type on DEV reset.

GAMW Power Context Save

PWRCTXSAVE - GAMW Power Context Save

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04000h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**Extra Bits15** Default Value: 0b Access: R/W Extra Bits for future use.
	14	**Extra Bits14** Default Value: 0b Access: R/W Extra Bits for future use.
	13	**Extra Bits13** Default Value: 0b Access: R/W Extra Bits for future use.
	12	**Extra Bits12** Default Value: 0b Access: R/W Extra Bits for future use.
	11	**Extra Bits11** Default Value: 0b Access: R/W Extra Bits for future use.

PWRCTXSAVE - GAMW Power Context Save

	10	**Extra Bits10** Default Value: 0b Access: R/W Extra Bits for future use.
	9	**Power Context Save Request** Default Value: 0b Access: R/W Power Context Save Bit[9] Power Context Save Request 1'b0: Power context save is not being requested (default). 1'b1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. This bit is self clear.
	8:0	**Power Context Save Quad Word Credits** Default Value: 000000000b Access: R/W Power Context Save Bits[8:0] QWord Credits for Power Context Save Request An initial length packet is required per power context save session, but that packet does not consume a credit. See protocol description for more details. Minimum Credits = 1: Unit may send 1 QWord pair. Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Only valid with PWRCTX_SAVE_REQ (Bit9).

Gather Constants Not Consumed By RCS

GATHER_CONST_PRODUCE_COUNT - Gather Constants Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	0248Ch

This register keeps track of the outstanding Gather Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**Gather Constants Produce Count** This register keeps track of the outstanding Gather Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

GDR Per Client Write Drop Enables

WR_DROP_MODE - GDR Per Client Write Drop Enables

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	040B4h

DWord	Bit	Description
0	31:0	**GDR Per Client Write Drop Enables** Default Value: 00000000h Access: R/W 31 RSVD: Future use. 30 MBC write drop disable (0) or enable (1). 29 CS write drop disable (0) or enable (1). 28 SOL write drop disable (0) or enable (1). 27 RS write drop disable (0) or enable (1). 26 RCC write drop disable (0) or enable (1). 25 MSC write drop disable (0) or enable (1). 24 All L3 clients write drop disable (0) or enable (1). 23 STC write drop disable (0) or enable (1). 22 HIZ write drop disable (0) or enable (1). 21 RCZ write drop disable (0) or enable (1). 20 GAFS write drop disable (0) or enable (1). 19 GPM write drop disable (0) or enable (1). 18 GCP write drop disable (0) or enable (1). 17 VCS write drop disable (0) or enable (1). 16 BSP write drop disable (0) or enable (1). 15 VCR write drop disable (0) or enable (1). 14 VMX_RS write drop disable (0) or enable (1). 13 VMX_BS write drop disable (0) or enable (1). 12 VMX_RA write drop disable (0) or enable (1). 11 VMX_VDS write drop disable (0) or enable (1). 10 VLF_RS write drop disable (0) or enable (1). 9 VLF_FW write drop disable (0) or enable (1). 8 VECS write drop disable (0) or enable (1). 7 VEO write drop disable (0) or enable (1). 5 uC (DMA) write drop disable (0) or enable (1). 4 BCS write drop disable (0) or enable (1). 3 BLB write drop disable (0) or enable (1). 2 W_BSP write drop disable (0) or enable (1). 1 W_VMX_RS write drop disable (0) or enable (1). 0 W_VMX_BS write drop disable (0) or enable (1).

GDR Per Client Write Drop Enables-2

WR_DROP_MODE2 - GDR Per Client Write Drop Enables-2		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	040B8h	
DWord	**Bit**	**Description**
0	31:0	**GDR Per Client Write Drop Enables-2** Default Value: 00000000h Access: R/W 31:23 RSVD: Future use. 22 BLB2 write drop disable (0) or enable (1). 21 HLF_FW write drop disable (0) or enable (1). 20 HLF_RS write drop disable (0) or enable (1). 19 HPP write drop disable (0) or enable (1). 18 HUC_DMA write drop disable (0) or enable (1). 17 HHI_IndirectHeader write drop disable (0) or enable (1). 16 HHI_Streamout write drop disable (0) or enable (1). 15 HHI_Bitstream write drop disable (0) or enable (1). 14 MFL_VLF_FW write drop disable (0) or enable (1). 13 MFL_VLF_RS write drop disable (0) or enable (1). 12 MFL_VMX_RA write drop disable (0) or enable (1). 11 MFL_VMX_RS write drop disable (0) or enable (1). 10 VD_ENC_SO write drop disable (0) or enable (1). 9 VD_ENC_RS write drop disable (0) or enable (1). 8 SFC_FW_VE write drop disable (0) or enable (1). 7 SFC_LB_VE write drop disable (0) or enable (1). 6 SFC_FW_VD write drop disable (0) or enable (1). 5 SFC_LB_VD write drop disable (0) or enable (1). 4 VOP write drop disable (0) or enable (1). 3 VDS write drop disable (0) or enable (1). 2 OA write drop disable (0) or enable (1). 1 VFE write drop disable (0) or enable (1). 0 VF write drop disable (0) or enable (1).

GDR Write Drop

GDR_WR_DRP - GDR Write Drop		
Register Space:	MMIO: 0/2/0	
Default Value:	0x80000000	
Size (in bits):	32	
Address:	04020h	

DWord	Bit	Description
0	31:0	**GDR_WRITE_DROP** Access: R/W Value: 80000000h — Name: **[Default]**

General Purpose Register

CS_GPR - General Purpose Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	1024
Address:	02600h-0267Fh
Name:	General Purpose Register
ShortName:	CS_GPR_RCSUNIT
Address:	12600h-1267Fh
Name:	General Purpose Register
ShortName:	CS_GPR_VCSUNIT0
Address:	1A600h-1A67Fh
Name:	General Purpose Register
ShortName:	CS_GPR_VECSUNIT
Address:	1C600h-1C67Fh
Name:	General Purpose Register
ShortName:	CS_GPR_VCSUNIT1
Address:	22600h-2267Fh
Name:	General Purpose Register
ShortName:	CS_GPR_BCSUNIT

Description	Source
This is a General Purpose Register bank of sixteen 64bit registers, which will be used as temporary storage by MI_MATH command to do ALU operations.	

GPR Index	MMIO Offset	Source
R_0	0x2600	RenderCS
R_1	0x2608	
R_2	0x2610	
R_3	0x2618	
R_4	0x2620	

CS_GPR - General Purpose Register

R_5	0x2628
R_6	0x2630
R_7	0x2638
R_8	0x2640
R_9	0x2648
R_10	0x2650
R_11	0x2658
R_12	0x2660
R_13	0x2668
R_14	0x2670
R_15	0x2678

DWord	Bit	Description
0	63:32	**CS_GPR_DATA1** Source: CommandStreamer
	31:0	**CS_GPR_DATA0** Source: CommandStreamer
1	63:32	**CS_GPR_DATA3** Source: CommandStreamer
	31:0	**CS_GPR_DATA2** Source: CommandStreamer
2	63:32	**CS_GPR_DATA5** Source: CommandStreamer
	31:0	**CS_GPR_DATA4** Source: CommandStreamer
3	63:32	**CS_GPR_DATA7** Source: CommandStreamer
	31:0	**CS_GPR_DATA6** Source: CommandStreamer
4	63:32	**CS_GPR_DATA9** Source: CommandStreamer
	31:0	**CS_GPR_DATA8** Source: CommandStreamer
5	63:32	**CS_GPR_DATA11** Source: CommandStreamer

CS_GPR - General Purpose Register		
	31:0	**CS_GPR_DATA10** Source: CommandStreamer
6	63:32	**CS_GPR_DATA13** Source: CommandStreamer
	31:0	**CS_GPR_DATA12** Source: CommandStreamer
7	63:32	**CS_GPR_DATA15** Source: CommandStreamer
	31:0	**CS_GPR_DATA14** Source: CommandStreamer
8	63:32	**CS_GPR_DATA17** Source: CommandStreamer
	31:0	**CS_GPR_DATA16** Source: CommandStreamer
9	63:32	**CS_GPR_DATA19** Source: CommandStreamer
	31:0	**CS_GPR_DATA18** Source: CommandStreamer
10	63:32	**CS_GPR_DATA21** Source: CommandStreamer
	31:0	**CS_GPR_DATA20** Source: CommandStreamer
11	63:32	**CS_GPR_DATA23** Source: CommandStreamer
	31:0	**CS_GPR_DATA22** Source: CommandStreamer
12	63:32	**CS_GPR_DATA25** Source: CommandStreamer
	31:0	**CS_GPR_DATA24** Source: CommandStreamer
13	63:32	**CS_GPR_DATA27** Source: CommandStreamer
	31:0	**CS_GPR_DATA26** Source: CommandStreamer

CS_GPR - General Purpose Register		
14	63:32	**CS_GPR_DATA29** Source: CommandStreamer
	31:0	**CS_GPR_DATA28** Source: CommandStreamer
15	63:32	**CS_GPR_DATA31** Source: CommandStreamer
	31:0	**CS_GPR_DATA30** Source: CommandStreamer

GFX Arbiter Client Priority Control

GFX_PRIO_CTRL - GFX Arbiter Client Priority Control

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x880A2D10
Size (in bits):	32
Address:	04A00h

DWord	Bit	Description
0	31:27	**Read Rstrm Max Reject** Default Value: 10001b Access: R/W
	26:21	**Extra Bits** Default Value: 000000b Access: R/W
	20:18	**sol_gam_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [18] is NOT Used.
	17:15	**veo_gam_priority** Default Value: 100b Access: R/W Client Priority Control Bits - Lowest Bit [15] is NOT Used.
	14:12	**vfw_gam_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [12] is NOT Used.
	11:9	**gapc_gam_c_priority** Default Value: 110b Access: R/W Client Priority Control Bits - Lowest Bit [9] is NOT Used.
	8:6	**gapc_gam_z_priority** Default Value: 100b Access: R/W Client Priority Control Bits - Lowest Bit [6] is NOT Used.

GFX_PRIO_CTRL - GFX Arbiter Client Priority Control

	5:3	**gapc_gam_l3_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [3] is NOT Used.
	2:0	**csrsvf_gam_priority** Default Value: 000b Access: R/W Client Priority Control Bits - Lowest Bit [0] is NOT Used.

GFX Context Element Descriptor (High Part)

GFX_CTX_EDR_H - GFX Context Element Descriptor (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04404h	
DWord	**Bit**	**Description**
0	31:0	**GFX Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W Bit[63:32] - Context ID: Context identification number assigned to separate this context from others. Context IDs need to be recycled in such a way that there cannot be two active contexts with the same ID. This is a unique identification number by which a context is identified and referenced.

GFX Context Element Descriptor (Low Part)

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Default Value:	0x00000009
Size (in bits):	32
Address:	04400h

DWord	Bit	Description
0	31:0	**GFX Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W Bit[31:12] - LRCA: Command Streamer Only. Bit[8] - Privileged Context / GGTT vs PPGTT mode: Differs in legacy vs advanced context modes: In Legacy Context: Defines the page tables to be used. This is how page walker comes to know PPGTT vs GGTT selection for the entire context. 0: Use Global GTT. 1: Use Per-Process GTT. In Advanced Context: Defines the privilege level for the context. 0: User mode context. 1: Supervisor mode context. Bit[5] - Deeper IA coherency Support: In Advanced Context: Defines the level of IA coherency. 0: IA coherency is provided at LLC level for all streams of GPU (i.e. Gen7.5 like mode). 1: IA coherency is provided at L3 level for EU data accesses of GPU. Bit[4] - A and D Support / 32 and 64b Address Support: Differs in legacy vs advanced context modes: In Legacy Context: Defines 32b vs 64b (48b canonical) addressing format: 0: 32b addressing format. 1: 64b (48b canonical) addressing format. In Advanced Context: Defines A and D bit support: 0: A and D bit management in page tables is NOT supported. 1: A and D bit management in page tables is supported. Bit[3] - Context Type: Legacy vs Advanced: Defines the context type 0: Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models). Note that advanced context is not bounded to GPGPU. 1: Legacy Context: Defines the context as legacy mode which is similar to prior generations of Gen8. Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected. Bit[2] - FR: Command streamer specific.

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)		
		Bit[1] - Scheduling Mode: 1: Indicates execlist mode of scheduling. 0: Indicates Ring Buffer mode of scheduling. Bit[0] - Valid: Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it continues but flags an error.

GFX Fault Counter

GFX_FAULT_CNTR - GFX Fault Counter

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	045A0h

DWord	Bit	Description
0	31:0	**GFX Fault Counter** Default Value: 00000000h Access: RO This counter only applies to advance context when fault and stream mode is selected.

GFX Fixed Counter

GFX_FIXED_CNTR - GFX Fixed Counter		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	045A4h	
DWord	**Bit**	**Description**
0	31:0	**GFX Fixed Counter** Default Value: 00000000h Access: RO This counter only applies to advance context when fault and stream mode is selected.

GFX PDP0/PML4/PASID Descriptor (High Part)

GFX_CTX_PDP0_H - GFX PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0440Ch

DWord	Bit	Description
0	31:0	**GFX PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W PDP0/PML4/PASID: This register can contain three values which depend on the element descriptor definition. PASID[19:0]: Populated in the first 20bits of the register and selected when Advanced Context flag is set. PML4[38:12]: Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected. PDP0[38:12]: Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping. Note: This is a guest physical address.

GFX PDP0/PML4/PASID Descriptor (Low Part)

GFX_CTX_PDP0_L - GFX PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	04408h

DWord	Bit	Description
0	31:0	**GFX PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W PDP0/PML4/PASID: This register can contain three values which depend on the element descriptor definition. PASID[19:0]: Populated in the first 20 bits of the register and selected when Advanced Context flag is set. PML4[38:12]: Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected. PDP0[38:12]: Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping. Note: This is a guest physical address.

GFX PDP1 Descriptor Register (High Part)

GFX_CTX_PDP1_H - GFX PDP1 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04414h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping. Note: This is a guest physical address.

GFX PDP1 Descriptor Register (Low Part)

GFX_CTX_PDP1_L - GFX PDP1 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04410h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping. Note: This is a guest physical address.

GFX PDP2 Descriptor Register (High Part)

GFX_CTX_PDP2_H - GFX PDP2 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0441Ch	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping. Note: This is a guest physical address.

GFX PDP2 Descriptor Register (Low Part)

GFX_CTX_PDP2_L - GFX PDP2 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04418h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping. Note: This is a guest physical address.

GFX PDP3 Descriptor Register (High Part)

GFX_CTX_PDP3_H - GFX PDP3 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04424h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping. Note: This is a guest physical address.

GFX PDP3 Descriptor Register (Low Part)

GFX_CTX_PDP3_L - GFX PDP3 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04420h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping. Note: This is a guest physical address.

Global Invalidation Register

GLBLINVL - Global Invalidation Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B404h	
DWord	**Bit**	**Description**
0	31:10	**Reserved** Access: RO Reserved.
	9:3	**Reserved**
	2	**Cross sync read disable** Access: R/W Ipconf_crs_sync_dis: Cross Sync Read Disable (CSRD). Cross Sync Read Disable (CSRD): Cross Sync Read Disable: upon a SYNC from HDC, follow with a write to cross SYNC Push and read to the same address. When set read is disabled. This bit needs to be programmed to 1 to disable fence operation for GSYNC. This is a workaround for GSYNC fence cycle encountering page fault.
	1	**Disables hashing function** Access: R/W Disables hashing function (DISHHF): Disables hashing function to generate bank_id[1:0] for L3$ bank accessing, and forces the use of address[7:6] for bank_id[1:0]. 0: (default) Hash function enabled to generate L3$ bank IDs. 1: L3$ address[7:6] used as L3$ bank IDs. Ipconf_csr_l3bankidhashdis. (This bit needs to set corresponding bit lncf_csr_l3bankidhashdis in LNCF.)
	0	**Reserved**

Global System Interrupt Routine

EU_GLOBAL_SIP - Global System Interrupt Routine

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E42Ch

DWord	Bit	Description
0	31:3	**Global SIP** Format: GraphicsAddress[31:3] Specifies the base address for System Interrupt Routine that over-rides the SIP set by the state (STATE_SIP).
	2:1	**Reserved** Format: PBC
	0	**Global SIP Enable** The bit specifies if the System Routine starts from the Global SIP provided by the DW OR the SIP provided by the state (STATE_SIP) Value / Name: 0 – SIP used is from STATE_EIP 1 – SIP used is from MMIO register

GMBUS0

GMBUS0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	C5100h-C5103h
Name:	GMBUS0 Clock/Port Select
ShortName:	GMBUS0
Power:	Always on
Reset:	soft

This register controls the clock rate of the serial bus and the device the controller is connected to.
This register should be configured before the first data valid bit is set.

DWord	Bit	Description
0	31:12	**Reserved** Format: MBZ
	11	**Reserved**
	10:8	**GMBUS Rate Select** This field selects the rate that the GMBUS will run at. It also defines the AC timing parameters used. Value / Name: 000b: 100 KHz 001b: 50 KHz Others: Reserved **Restriction** Restriction : It should only be changed between transfers when the GMBUS is idle.
	7	**Reserved** Format: MBZ
	6	**Byte Count Override** **Description** This field overrides the byte count to allow burst reads of greater than 511 bytes. See the GMBUS and GPIO page for programming instructions. Field not valid until KBLPCH-H A0, SPT-LP D1, SPT-H E1

GMBUS0

		Value / **Name** 0b / Disable 1b / Enable
	5	**Reserved** Format: MBZ
	4:3	**Reserved** Format: MBZ
	2:0	**Pin Pair Select** This field selects a GMBUS pin pair for use in the GMBUS communication. See the table of GPIO Pin Usages to determine which pin pairs are supported and their intended functions.

Value	Name
000b	None (Disabled)
100b	DDIC
101b	DDIB
110b	DDID

GMBUS1

GMBUS1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W Protect
Size (in bits):	32
Address:	C5104h-C5107h
Name:	GMBUS1 Command/Status
ShortName:	GMBUS1
Power:	Always on
Reset:	soft

This register lets the software indicate to the GMBUS controller the slave device address, register index, and indicate when the data write is complete.
When the SW_CLR_INT bit is asserted, all writes to the GMBUS2, GMBUS3, and GMBUS4 registers are discarded. The GMBUS1 register writes to any other bit except the SW_CLR_INT are also lost.
Reads to these registers always work normally regardless of the state of the SW_CLR_INT bit.

DWord	Bit	Description
0	31	**Software Clear Interrupt** Access: R/W (SW_CLR_INT) This bit must be clear for normal operation. Setting the bit then clearing it acts as local reset to the GMBUS controller. This bit is commonly used by software to clear a BUS_ERROR when a slave device delivers a NACK. **Value / Name / Description** 0b / Clear HW_RDY / If this bit is written as a zero when its current state is a one, will clear the HW_RDY bit and allows register writes to be accepted to the GMBUS registers (Write Protect Off). This bit is cleared to zero when an event causes the HW_RDY bit transition to occur. 1b / Assert HW_RDY / Asserted by software after servicing the GMBUS interrupt. Setting this bit causes the INT status bit to be cleared. Setting (1) this bit also asserts the HW_RDY bit (until this bit is written with a 0). When this bit is set, no writes to GMBUS registers will cause the contents to change with the exception of this bit which can be written.
	30	**Software Ready** (SW_RDY) Data handshake bit used in conjunction with HW_RDY bit. **Value / Name / Description** 0b / De-Assert / De-asserted via the assertion event for HW_RDY bit

GMBUS1

1b	SW Assert	When asserted by software, results in de-assertion of HW_RDY bit

29 Enable Timeout

(ENT) Enables timeout for slave response.

When this bit is enabled and the slave device response has exceeded the timeout period, the GMBUS Slave Stall Timeout Error interrupt bit is set.

Value	Name
0b	Disable
1b	Enable

28 Reserved

Format:	MBZ

27:25 Bus Cycle Select

GMBUS cycle will always consist of a START followed by Slave Address, followed by an optional read or write data phase. A read cycle with an index will consist of a START followed by a Slave Address a WRITE indication and the INDEX and then a RESTART with a Slave Address and an optional read data phase. The GMBUS cycle will terminate either with a STOP or by entering a wait state. The WAIT state is exited by generating a STOP or by starting another GMBUS cycle. This can only cause a STOP to be generated if a GMBUS cycle is generated, the GMBUS is currently in a data phase, or it is in a WAIT phase.

The three bits can be decoded as follows:

27 = STOP generated

26 = INDEX used

25 = Cycle ends in a WAIT

Value	Name	Description
000b	No cycle	No GMBUS cycle is generated
001b	No Index, No Stop, Wait	GMBUS cycle is generated without an INDEX, with no STOP, and ends with a WAIT
010b	Reserved	Reserved
011b	Index, No Stop, Wait	GMBUS cycle is generated with an INDEX, with no STOP, and ends with a WAIT
100b	Gen Stop	Generates a STOP if currently in a WAIT or after the completion of the current byte if active
101b	No Index, Stop	GMBUS cycle is generated without an INDEX and with a STOP
110b	Reserved	Reserved
111b	Index, Stop	GMBUS cycle is generated with an INDEX and with a STOP

24:16 Total Byte Count

This determines the total number of bytes to be transferred during the DATA phase of a GMBUS cycle.

The DATA phase can be prematurely terminated by generating a STOP while in the DATA phase (see Bus Cycle Select).

Restriction

GMBUS1

		Restriction : Do not change the value of this field during GMBUS cycles transactions. The byte count must not be zero.
	15:8	**8 bit Slave Register Index** (INDEX) This field specifies the 8-bits of index to be used for the generated bus write transaction or the index used for the WRITE portion of the WRITE/READ pair. It only has an effect if the enable Index bit is set. **Restriction** Restriction : Do not change the value of this field during GMBUS cycles transactions.
	7:0	**Slave Address And Direction** Bits 7:1 = 7-bit GMBUS Slave Address (SADDR): When a GMBUS cycle is to be generated using the Bus Cycle Select field, this field specifies the value of the slave address that is to be sent out. For use with 10-bit slave address devices, set this value to 11110XXb (where the last two bits (XX) are the two MSBs of the 10-bit address) and the slave direction bit to a write. This is followed by the first data byte being the 8 LSBs of the 10-bit slave address. Bit 0 = Slave Direction Bit: When a GMBUS cycle is to be generated based on the Bus Cycle Select, this bit determines if the operation will be a read or a write. A read operation with the index enabled will perform a write with just the index followed by a re-start and a read. A 1 indicates that a Read from the slave device operation is to be performed. A 0 indicates that a Write to the slave device operation is to be performed.

Value	Name
00000001b	General Call Address
00000000b	Start Byte
0000001Xb	CBUS Address
11110XXXb	10-Bit Addressing
Others	Reserved

GMBUS2

GMBUS2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000800
Access:	R/W Protect
Size (in bits):	32
Address:	C5108h-C510Bh
Name:	GMBUS2 Status
ShortName:	GMBUS2
Power:	Always on
Reset:	soft

When the SW_CLR_INT bit is asserted, all writes to this registet are discarded. Reads to this register always work normally regardless of the state of the SW_CLR_INT bit.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15	**INUSE** Software wishing to arbitrate for the GMBUS resource can poll this bit until it reads a zero and will then own usage of the GMBUS controller. This bit has no effect on the hardware, and is only used as semaphore among various independent software threads. Writing a one to this bit is software's indication that the software use of this resource is now terminated and it is available for other clients.
	14	**Hardware Wait Phase** Access: RO Once in a WAIT_PHASE, the software can now choose to generate a STOP cycle or a repeated start (RESTART) cycle followed by another GMBUS transaction. Wait phase is entered at the end of the current transaction when that transaction is selected not to terminate with a STOP.

Value	Name	Description
0b	GMBUS is Acquired	Read operation that contains a zero in this bit position indicates that the GMBUS engine is now acquired and the subsequent reads of this register will now have this bit set. Writing a 0 to this bit has no effect.
1b	GMBUS in Use	Read operation that contains a one for this bit indicates that the GMBUS is currently allocated to someone else and "In use". Once set, a write of a 1 to this bit indicates that the software has relinquished the GMBUS resource and will reset the value of this bit to a 0.

GMBUS2

Value	Name
0b	Not in a wait phase
1b	In wait phase

13 **Slave Stall Timeout Error**

Access:	RO

This bit indicates that a slave stall timeout has occurred. It is tied to the Enable Timeout (ENT) bit.

Value	Name
0b	No Slave Timeout
1b	Slave Timeout

12 **GMBUS Interrupt Status**

Access:	RO

This bit indicates that an event that causes a GMBUS interrupt has occurred. The interrupt can be caused by one of the interrupt types enabled in the GMBUS4 register

Value	Name
0b	No Interrupt
1b	Interrupt

11 **Hardware Ready**

Access:	RO

(HW_RDY) This provides a method of detecting when the current software client routine can proceed with the next step in a sequence of GMBUS operations.
This data handshake bit is used in conjunction with the SW_RDY bit.
When this bit is asserted by the GMBUS controller, it results in the de-assertion of the SW_RDY bit.
This bit resumes to normal operation when the SW_CLR_INT bit is written to a 0.

Value	Name	Description
0b	0	Condition required for assertion has not occurred or when this bit is a one and: - SW_RDY bit has been asserted - During a GMBUS read transaction, after the each read of the data register - During a GMBUS write transaction, after each write of the data register - SW_CLR_INT bit has been cleared
1b	1 **[Default]**	This bit is asserted under the following conditions: - After a reset or when the transaction is aborted by the setting of the SW_CLR_INT bit - When an active GMBUS cycle has terminated with a STOP - When during a GMBUS write transaction, the data register needs and can accept another four bytes of data - During a GMBUS read transaction, this bit is asserted when the data register has four bytes of new data or the read transaction DATA phase is

GMBUS2

		complete and the data register contains the last few bytes of the read data

10 NAK Indicator

Access:	RO

MAK is indicated by hardware if any expected device acknowledge is not received from the slave within the timeout.

Value	Name
0b	No bus error
1b	NAK occurred

9 GMBUS Active

Access:	RO

(GA) This is a status bit that indicates whether the GMBUS controller is in an IDLE state or not. Active states are the START, ADDRESS, INDEX, DATA, WAIT, or STOP Phase.

Value	Name
0b	Idle
1b	Active

8:0 Current Byte Count

Access:	RO

Can be used to determine the number of bytes currently transmitted/received by the GMBUS controller hardware.

Hardware sets it to zero at the start of a GMBUS transaction data transfer and incremented after the completion of each byte of the data phase.

Note that because reads have internal storage, the byte count on a read operation may be ahead of the data that has been accepted from the data register.

GMBUS3

GMBUS3	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W Protect
Size (in bits):	32
Double Buffer Update Point:	HW_RDY
Address:	C510Ch-C510Fh
Name:	GMBUS3 Data Buffer
ShortName:	GMBUS3
Power:	Always on
Reset:	soft

This is the data read/write register. This register is double buffered.
Bit 0 is the first bit sent or read, bit 7 is the 8th bit sent or read, all the way through bit 31 being the 32nd bit sent or read.
For GMBUS write operations with a non-zero byte count, this register should be written with the data before the GMBUS cycle is initiated.
For byte counts that are greater than four bytes, this register will be written with subsequent data only after the HW_RDY status bit is set indicating that the register is now ready for additional data.
For GMBUS read operations, software should wait until the HW_RDY bit indicates that the register contains the next set of valid read data before reading this register.
When the SW_CLR_INT bit is asserted, all writes to this registet are discarded. Reads to this register always work normally regardless of the state of the SW_CLR_INT bit.

DWord	Bit	Description
0	31:24	**Data Byte 3**
	23:16	**Data Byte 2**
	15:8	**Data Byte 1**
	7:0	**Data Byte 0**

GMBUS4

GMBUS4	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W Protect
Size (in bits):	32
Address:	C5110h-C5113h
Name:	GMBUS4 Interrupt Mask
ShortName:	GMBUS4
Power:	Always on
Reset:	soft

When the SW_CLR_INT bit is asserted, all writes to this registet are discarded. Reads to this register always work normally regardless of the state of the SW_CLR_INT bit.

DWord	Bit	Description
0	31:5	**Reserved** Format: MBZ
	4:0	**Interrupt Mask** This field specifies which GMBUS interrupts events may contribute to the setting of GMBUS interrupt status bit in the second level interrupt status register. For writes, the HW Ready (HWRDY) interrupt indicates that software can write the next DWORD. It does NOT mean that the transfer of data to the slave device has completed. The IDLE or HW wait interrupt may be used to detect the end of writing data to the slave device. The HWRDY interrupt may be used for gmbus write cycles only to detect when to write the next DWORD after the first two DWORDs have been written to GMBUS3. For reads, the HWRDY interrupt indicates the arrival of the next dword.

Value	Name
0XXXXb	Slave Stall Timeout Interrupt Disable
1XXXXb	Slave Stall Timeout Interrupt Enable
X0XXXb	NAK Interrupt Disable
X1XXXb	NAK Interrupt Enable
XX0XXb	Idle Interrupt Disable
XX1XXb	Idle Interrupt Enable
XXX0Xb	HW Wait Interrupt (cycle without a stop has completed) Disable
XXX1Xb	W Wait Interrupt (cycle without a stop has completed) Enable
XXXX0b	HW Ready (Data transferred) Interrupt Disable
XXXX1b	HW Ready (Data transferred) Interrupt Enable

GMBUS5

GMBUS5		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	C5120h-C5123h	
Name:	GMBUS5 2 Byte Index	
ShortName:	GMBUS5	
Power:	Always on	
Reset:	soft	
This register provides a method for the software indicate to the GMBUS controller the 2 byte device index.		
DWord	**Bit**	**Description**
0	31	**2 Byte Index Enable** When this bit is asserted (1), then bits 15:0 are used as the index. Bits 15:8 are used in the first byte which is the most significant index bits. The slave index in the GMBUS1<15:8> are ignored. Bits 7:0 are used in the second byte which is the least significant index bits.
	30:16	**Reserved** Format: MBZ
	15:0	**2 Byte Slave Index** This is the 2 byte index used in all GMBUS accesses when bit 31 is asserted (1).

GMCH Graphics Control

GGC_0_0_0_PCI - GMCH Graphics Control

Register Space:	PCI: 0/0/0
Source:	BSpec
Default Value:	0x00000500
Size (in bits):	16
Address:	00050h

All the bits in this register are LT lockable.

DWord	Bit	Description
0	15:8	**Graphics Mode Select** Default Value: 00000101b Access: R/W Lock This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled. BIOS Requirement: BIOS must not set this field to 0h if IVD (bit 1 of this register) is 0. BIOS Requirement: Given new sizes allow down to 8MB allocation, BIOS has to ensure there is sufficient space for WOPCM and basic GFX Stolen functions. 00h:0MB 01h:32MB 02h:64MB 03h:96MB 04h:128MB 05h:160MB (default) 06h:192MB 07h:224MB 08h:256MB 09h:288MB 0Ah:320MB 0Bh:352MB 0Ch:384MB 0Dh:416MB 0Eh:448MB 0Fh:480MB 10h:512MB 11h - 1Fh: Reserved 20h:1024MB 21h - 2Fh: Reserved 30h:1536MB 31h - 3Fh: Reserved 40h: 2048MB

GGC_0_0_0_PCI - GMCH Graphics Control

		41h - EFh: Reserved F0h: 4MB F1h: 8MB F2h: 12MB F3h: 16MB F4h: 20MB F5h: 24MB F6h: 28MB F7h: 32MB F8h: 36MB F9h: 40MB FAh: 44MB FBh: 48MB FCh: 52MB FDh: 56MB FEh: 60MB FFh: Reserved Hardware functionality in case of programming this value to Reserved is not guaranteed.
	7:6	**GTT Graphics Memory Size** Default Value: 00b Access: R/W Lock This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics Translation Table. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. GSM is assumed to be a contiguous physical DRAM space with DSM, and BIOS needs to allocate a contiguous memory chunk. Hardware will derive the base of GSM from DSM only using the GSM size programmed in the register. Hardware functionality in case of programming this value to Reserved is not guaranteed. 0x0: No Preallocated Memory 0x1: 2MB of Preallocated Memory 0x2: 4MB of Preallocated Memory 0x3: 8MB of Preallocated Memory
	5:3	**Reserved** Format: MBZ
	2	**Versatile Acceleration Mode Enable** Default Value: 0b Access: R/W Lock Enables the use of the iGFX engines for Versatile Acceleration. 0 - iGFX engines are in iGFX Mode. Device 2 Class Code is 030000h. 1 - iGFX engines are in Versatile Acceleration Mode. Device 2 Class Code is 048000h.

GGC_0_0_0_PCI - GMCH Graphics Control

	1	**IGD VGA Disable** Default Value: 0b Access: R/W Lock 0: Enable. Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00. 1: Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub- Class Code field within Device 2 function 0 Class Code register is 80. BIOS Requirement: BIOS must not set this bit to 0 if the GMS field pre-allocates no memory. This bit MUST be set to 1 if Device 2 is disabled either via a fuse or fuse override (CAPID0_A[IGD] = 1) or via a register (DEVEN[3] = 0).
	0	**GGC Lock** Default Value: 0b Access: R/W Key Lock When set to 1b, this bit will lock all bits in this register.

Go Protocol GAM Request

GO_GAM_REQ - Go Protocol GAM Request

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000 [KBL]
Size (in bits):	32
Address:	040D0h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Reserved.
	15	**GO_PROTOCOL_GAM_REQUEST15** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for FLR (device) reset (cdevrst_b).
	14	**GO_PROTOCOL_GAM_REQUEST14** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Media1 reset (vcs1unit).
	13	**GO_PROTOCOL_GAM_REQUEST13** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Wi-Di reset (winunit).
	12	**GO_PROTOCOL_GAM_REQUEST12** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset).

GO_GAM_REQ - Go Protocol GAM Request

	11	**GO_PROTOCOL_GAM_REQUEST11** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Blitter reset (bcsunit).
	10	**GO_PROTOCOL_GAM_REQUEST10** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for VEBox reset (vecsunit).
	9	**GO_PROTOCOL_GAM_REQUEST9** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Media0 reset (vcs0unit).
	8	**GO_PROTOCOL_GAM_REQUEST8** Default Value: 0b Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Render reset (csunit).
	7	**GO_PROTOCOL_GAM_REQUEST7** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls OA Cycles (oaunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.

GO_GAM_REQ - Go Protocol GAM Request

	6	**GO_PROTOCOL_GAM_REQUEST6** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Wi-Di Cycles (winunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	5	**Reserved**
	4	**GO_PROTOCOL_GAM_REQUEST4** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Blitter Cycles (bcsunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	3	**GO_PROTOCOL_GAM_REQUEST3** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls VEBox Cycles (vecsunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	2	**GO_PROTOCOL_GAM_REQUEST2** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Media1 Cycles (vcs1unit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.

GO_GAM_REQ - Go Protocol GAM Request

	1	**GO_PROTOCOL_GAM_REQUEST1** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Media0 Cycles (vcs0unit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	0	**GO_PROTOCOL_GAM_REQUEST0** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Render Cycles (csunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.

GPA to HPA Translation Request

GPA2HPAR - GPA to HPA Translation Request

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0420Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**GPA to HPA Translation Request 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**GPA to HPA Translation Request 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**GPA to HPA Translation Request 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**GPA to HPA Translation Request 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GPA2HPAR - GPA to HPA Translation Request

	11	**GPA to HPA Translation Request 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	10	**GPA to HPA Translation Request 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**GPA to HPA Translation Request 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**GPA to HPA Translation Request 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**GPA to HPA Translation Request 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**GPA to HPA Translation Request 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GPA2HPAR - GPA to HPA Translation Request

	5	**GPA to HPA Translation Request 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**GPA to HPA Translation Request 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**GPA to HPA Translation Request 3** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	2	**GPA to HPA Translation Request 2** Default Value: 0b Access: R/W Bit[2]: A request for GPA to HPA translation. Note that GPA register should have been written prior to sending the message for the translation. Mask bit[18] needs to be enabled to program the register. This bit is self clear.
	1	**GPA to HPA Translation Request 1** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	0	**GPA to HPA Translation Request 0** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GPA value for GPA to HPA Translation

GPA2HPAV - GPA value for GPA to HPA Translation		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04210h	
DWord	**Bit**	**Description**
0	31:0	**GPA value for GPA to HPA Translation** Default Value: 00000000h Access: R/W The GPA value of the page that requires the GPA=>HPA translation bits[39:12] map to [28:1] of the register.

GPGPU Context Restore Request To TDL

GPGPU_CTX_RESTORE - GPGPU Context Restore Request To TDL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S0_SS0
Address:	0E5CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S0_SS1
Address:	0E6CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S0_SS2
Address:	0E4DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S1_SS0
Address:	0E5DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S1_SS1
Address:	0E6DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S1_SS2
Address:	0E4ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S2_SS0
Address:	0E5ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S2_SS1
Address:	0E6ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S2_SS2

GPGPU_CTX_RESTORE - GPGPU Context Restore Request To TDL

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

GPGPU Context Save Request To TDL

GPGPU_CTX_SAVE - GPGPU Context Save Request To TDL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4D8h

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

GPGPU Dispatch Dimension X

GPGPU_DISPATCHDIMX - GPGPU Dispatch Dimension X		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	02500h	
DWord	**Bit**	**Description**
0	31:0	**Dispatch Dimension X** Format: U32 The number of thread groups to be dispatched in the X dimension (max x + 1). **Value** / **Name** 0, FFFFFFFFh

GPGPU Dispatch Dimension Y

GPGPU_DISPATCHDIMY - GPGPU Dispatch Dimension Y

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02504h

DWord	Bit	Description
0	31:0	**Dispatch Dimension Y** Format: U32 The number of thread groups to be dispatched in the Y dimension (max y + 1

Value	Name
0, FFFFFFFFh	

GPGPU Dispatch Dimension Z

GPGPU_DISPATCHDIMZ - GPGPU Dispatch Dimension Z		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	02508h	

DWord	Bit	Description
0	31:0	**Dispatch Dimension Z** Format: U32 The number of thread groups to be dispatched in the Zdimension (max Z + 1) Value: 0, FFFFFFFFh — Name:

GPIO_CTL

GPIO_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000808
Access:	R/W
Size (in bits):	32
Address:	C501Ch-C501Fh
Name:	GPIO Control 3
ShortName:	GPIO_CTL_3
Power:	Always on
Reset:	soft
Address:	C5020h-C5023h
Name:	GPIO Control 4
ShortName:	GPIO_CTL_4
Power:	Always on
Reset:	soft
Address:	C5024h-C5027h
Name:	GPIO Control 5
ShortName:	GPIO_CTL_5
Power:	Always on
Reset:	soft

The register controls a pair of pins that can be used for general purpose control, but usually is designated for specific functions according to the requirements of the device and the system that the device is in.
Each pin of the two pin pair is designated as a clock or data for descriptive purposes. See the table at the beginning of this section to determine which pins/registers are supported and their intended functions.
Board design variations are possible and would affect the usage of these pins.
There are multiple instances of this register to support each of the GPIO pin pairs.

DWord	Bit	Description
0	31:13	**Reserved** Format: MBZ
	12	**GPIO Data In** Default Value: Ub Undefined (read only depends on I/O pin) Access: RO This is the value that is sampled on the GPIO_Data pin as an input. This bit is undefined at reset.

GPIO_CTL

11 GPIO Data Value

Default Value:	1b
Access:	R/W

This is the value that should be place on the GPIO Data pin as an output.
This value is only written into the register if GPIO DATA MASK is also asserted.
The value will appear on the pin if this data value is actually written to this register and the GPIO Data DIRECTION VALUE contains a value that will configure the pin as an output.
The default of '1' mimics the I2C external pull-ups.

10 GPIO Data Mask

Access:	WO

This is a mask bit to determine whether the GPIO DATA VALUE bit should be written into the register.

Value	Name
0b	Dot NOT write
1b	Write

9 GPIO Data Direction Value

Access:	R/W

This is the value that should be used to define the output enable of the GPIO Data pin.
This value is only written into the register if GPIO Data DIRECTION MASK is also asserted.
The value that will appear on the pin is defined by what is in the register for the GPIO DATA VALUE bit.

Value	Name
0b	Input
1b	Output

8 GPIO Data Direction Mask

Access:	WO

This is a mask bit to determine whether the GPIO Data DIRECTION VALUE bit should be written into the register.

Value	Name
0b	Dot NOT write
1b	Write

7:5 Reserved

Format:	MBZ

4 GPIO Clock Data In

Default Value:	Ub Undefined (read only depends on I/O pin)
Access:	RO

This is the value that is sampled on the GPIO Clock pin as an input.
This bit is undefined at reset.

GPIO_CTL

3 GPIO Clock Data Value

Default Value:	1b
Access:	R/W

This is the value that should be place on the GPIO Clk pin as an output.
This value is only written into the register if GPIO Clock DATA MASK is also asserted.
The value will appear on the pin if this data value is actually written to this register and the GPIO Clock DIRECTION VALUE contains a value that will configure the pin as an output.
The default of '1' mimics the I2C external pull-ups.

2 GPIO Clock Data Mask

Access:	WO

This is a mask bit to determine whether the GPIO Clock DATA VALUE bit should be written into the register.

Value	Name
0b	Dot NOT write
1b	Write

1 GPIO Clock Direction Value

Access:	R/W

This is the value that should be used to define the output enable of the GPIO Clock pin.
This value is only written into the register if GPIO Clock DIRECTION MASK is also asserted.
The value that will appear on the pin is defined by what is in the register for the GPIO Clock DATA VALUE bit.

Value	Name
0b	Input
1b	Output

0 GPIO Clock Direction Mask

Access:	WO

This is a mask bit to determine whether the GPIO Clock DIRECTION VALUE bit should be written into the register.

Value	Name
0b	Dot NOT write
1b	Write

GPM POWERGATE LICENSE REQUEST

GPM_POWERGATE_LICENSE_REQ - GPM POWERGATE LICENSE REQUEST

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00C0Ch

GPM-RPM PowerGate License Request
RPM Detects change in PowerGate License Request, forwards the request to PCU on C2U Event Bus

DWord	Bit	Description
0	31:0	**PowerGate License Request Data** Access: R/W 31:20 : Not used 19:16 : Render Slice Count 15:11 : Subslice Count 10:8 : Media Count 7:0 : EU Count

GP Thread Time

GP_THREAD_TIME - GP Thread Time	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	054C4h
Name:	GP Thread Time
ShortName:	GP_THREAD_TIME

Reading this register returns the cumulative GP context execution time. This register uses the same clock frequency as CTX_TIMESTAMP, but differs from CTX_TIMESTAMP because it excludes the execution time during preemption save or restore. This register gets context save/restored on a context switch.
The granularity of this toggle is at the rate of the bit 3 in the "Reported Timestamp Count" register(0x2358). The toggle will be 8 times slower that "Reported Timestamp Count".

DWord	Bit	Description
0	31:0	**Timestamp Value** Access: RO Number of clock ticks that the context has run.

GPU_Ticks_Counter

GPU_TICKS - GPU_Ticks_Counter	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02910h

Reading this register returns the live value of the GPU ticks counter that is sampled and included in the performance counter report header. It increments every GFX clock and hence the number of increments per second changes with GFX frequency.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the GPU tick counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Graphics Device Reset Control

GDRST - Graphics Device Reset Control

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0941Ch

Graphics Device Reset Control Registers

DWord	Bit	Description
0	31:10	**Reserved** Access: RO Reserved
	9	**Initiate Graphics SFC1 soft reset** Access: R/W Set Graphics SFC 1 Soft-Reset Control: '1' : Initiate a graphics SFC1 domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP hardware can reset this bit. Note: This is a non-posted register.
	8	**Initiate Graphics SFC0 soft reset** Access: R/W Set Graphics SFC 0 Soft-Reset Control: '1' : Initiate a graphics SFC0 domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP hardware can reset this bit. Note: This is a non-posted register.
	7	**Initiate Graphics Media1 soft reset** Access: R/W Set Graphics Media 1 Soft-Reset Control: '1' : Initiate a graphics Vebox domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.

GDRST - Graphics Device Reset Control

	6	**Reserved**
	5	**Reserved**
	4	**Initiate Graphics Vebox Soft Reset** Access: R/W Set Graphics VEbox Soft-Reset Control: '1' : Initiate a graphics Vebox domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	3	**Initiate Graphics Blitter Soft Reset** Access: R/W Set Graphics Blitter Soft-Reset Control: '1' : Initiate a graphics blitter domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	2	**Initiate Graphics Media Soft Reset** Access: R/W Set Graphics Media Soft-Reset Control: '1' : Initiate a graphics media 0 domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	1	**Initiate Graphics Render Soft Reset** Access: R/W Set Graphics Render Soft-Reset Control: '1' : Initiate a graphics render domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.

GDRST - Graphics Device Reset Control

	0	**Initiate Graphics Full Soft Reset** Access: R/W Set Graphics Full Soft-Reset Control: '1' : Initiate a full graphics reset (that is, graphics render, media, and blitter reset). for GWL : '1' : Initiate a full graphics reset (that is, graphics media reset). - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.

Graphics Memory Fence Table Register

FENCE - Graphics Memory Fence Table Register	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	100000h-100007h
Name:	FENCE_0
Address:	100008h-10000Fh
Name:	FENCE_1
Address:	100010h-100017h
Name:	FENCE_2
Address:	100018h-10001Fh
Name:	FENCE_3
Address:	100020h-100027h
Name:	FENCE_4
Address:	100028h-10002Fh
Name:	FENCE_5
Address:	100030h-100037h
Name:	FENCE_6
Address:	100038h-10003Fh
Name:	FENCE_7
Address:	100040h-100047h
Name:	FENCE_8
Address:	100048h-10004Fh
Name:	FENCE_9
Address:	100050h-100057h
Name:	FENCE_10
Address:	100058h-10005Fh
Name:	FENCE_11
Address:	100060h-100067h
Name:	FENCE_12

FENCE - Graphics Memory Fence Table Register	
Address:	100068h-10006Fh
Name:	FENCE_13
Address:	100070h-100077h
Name:	FENCE_14
Address:	100078h-10007Fh
Name:	FENCE_15
Address:	100080h-100087h
Name:	FENCE_16
Address:	100088h-10008Fh
Name:	FENCE_17
Address:	100090h-100097h
Name:	FENCE_18
Address:	100098h-10009Fh
Name:	FENCE_19
Address:	1000A0h-1000A7h
Name:	FENCE_20
Address:	1000A8h-1000AFh
Name:	FENCE_21
Address:	1000B0h-1000B7h
Name:	FENCE_22
Address:	1000B8h-1000BFh
Name:	FENCE_23
Address:	1000C0h-1000C7h
Name:	FENCE_24
Address:	1000C8h-1000CFh
Name:	FENCE_25
Address:	1000D0h-1000D7h
Name:	FENCE_26
Address:	1000D8h-1000DFh
Name:	FENCE_27
Address:	1000E0h-1000E7h
Name:	FENCE_28
Address:	1000E8h-1000EFh

FENCE - Graphics Memory Fence Table Register

Name:	FENCE_29
Address:	1000F0h-1000F7h
Name:	FENCE_30
Address:	1000F8h-1000FFh
Name:	FENCE_31

The graphics device performs address translation from linear space to tiled space for a CPU access to graphics memory (See Memory Interface Functions chapter for information on these memory layouts) using the fence registers. Note that the fence registers are used only for CPU accesses to gfx memory. Graphics rendering/display pipelines use Per Surface Tiling (PST) parameters (found in SURFACE_STATE - see the Sampling Engine chapter) to access tiled gfx memory.

The intent of tiling is to locate graphics data that are close (in X and Y surface axes) in one physical memory page while still locating some amount of line oriented data sequentially in memory for display efficiency. All 3D rendering is done such that the QWords of any one span are all located in the same memory page, improving rendering performance. Applications view surfaces as linear, hence when the cpu access a surface that is tiled, the gfx hardware must perform linear to tiled address conversion and access the correct physical memory location(s) to get the data.

Tiled memory is supported for rendering and display surfaces located in graphics memory. A tiled memory surface is a surface that has a width and height that are subsets of the tiled region's pitch and height. The device maintains the constants required by the memory interface to perform the address translations. Each tiled region can have a different pitch and size. The CPU-memory interface needs the surface pitch and tile height to perform the address translation. It uses the GMAddr (PCI-BAR) offset address to compare with the fence start and end address, to determine if the rendering surface is tiled. The tiled address is generated based on the tile orientation determined from the matching fence register. Fence ranges are at least 4 KB aligned. Note that the fence registers are used only for CPU accesses to graphics memory.

A Tile represents 4 KB of memory. Tile height is 8 rows for X major tiles and 32 rows for Y major tiles. Tile Pitch is 512Bs for X major tiles and 128Bs for Y major tiles. The surface pitch is programmed in 128B units such that the pitch is an integer multiple of "tile pitch".

Engine restrictions on tile surface usage are detailed in Surface Placement Restrictions (Memory Interface Functions). Note that X major tiles can be used for Sampler, Color, Depth, motion compensation references and motion compensation destination, Display, Overlay, GDI Blt source and destination surfaces. Y major tiles can be used for Sampler, depth, color and motion compensation assuming they do not need to be displayed. GDI Blit operations, overlay and display cannot used Tiled Y orientations.

A "PST" graphics surface that will also be accessed via fence needs its base address to be tile row aligned.

Hardware handles the flushing of any pending cycles when software changes the fence upper/lower bounds.

Fence Table Registers occupy the address range specified above. Each Fence Table Register has the following format.

FENCE registers are not reset by a graphics reset. They will maintain their values unless a full reset is performed.

DWord	Bit	Description
0	63:44	**Fence Upper Bound** Format: GraphicsAddress[31:12] Bits 31:12 of the ending Graphics Address of the fence region. Fence regions must be aligned to a 4KB page. This address represents the last 4KB page of the fence region (Upper Bound is included in the fence region).

FENCE - Graphics Memory Fence Table Register

	Graphics Address is the offset within GMADR space.
43	**Reserved** Format: MBZ
42:32	**Fence Pitch** Format: U10-1 Width in 128 bytes This field specifies the width (pitch) of the fence region in multiple of "tile width". For Tile X this field must be programmed to a multiple of 512B ("003" is the minimum value) and for Tile Y this field must be programmed to a multiple of 128B ("000" is the minimum value). 000h = 128B 001h = 256B ... 3FFh = 128KB ... 7FFh = 256KB
31:12	**Fence Lower Bound** Format: GraphicsAddress[31:12] Bits 31:12 of the starting Graphics Address of the fence region. Fence regions must be aligned to 4KB. This address represents the first 4KB page of the fence region (Lowe Bound is included in the fence region). Graphics Address is the offset within GMADR space.
11:2	**Reserved** Format: MBZ
1	**Tile Walk** This field specifies the spatial ordering of QWords within tiles.

Value	Name	Description
0h	MI_TILE_XMAJOR	Consecutive SWords (32 Bytes) sequenced in the X direction
1h	MI_TILE_YMAJOR	Consecutive OWords (16 Bytes) sequenced in the Y direction

0	**Fence Valid** Format: MI_FenceValid This field specifies whether or not this fence register defines a fence region.

Value	Name
0h	MI_FENCE_INVALID
1h	MI_FENCE_VALID

Graphics Memory Range Address

GMADR_0_2_0_PCI - Graphics Memory Range Address

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x0000000C, 0x00000000
Size (in bits):	64
Address:	00018h

GMADR is the PCI aperture used by S/W to access tiled GFX surfaces in a linear fashion.

DWord	Bit	Description
0	63:39	**Reserved for Memory Base Address** Default Value: 000000000000000000000000b Access: R/W Must be set to 0 since addressing above 512GB is not supported.
	38:32	**Memory Base Address** Default Value: 0000000b Access: R/W Set by the OS, these bits correspond to address signals [38:32].
	31	**4096 MB Address Mask** Default Value: 0b Access: R/W Lock FLR Resettable This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO) depending on the value of MSAC.APSZ. RO and forced to 0 when MSAC.APSZ >= 4096MB. (i.e. MSAC.APSZ[4]=1)
	30	**2048 MB Address Mask** Default Value: 0b Access: R/W Lock FLR Resettable This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO) depending on the value of MSAC.APSZ. RO and forced to 0 when MSAC.APSZ >= 2048MB. (i.e. MSAC.APSZ[3]=1)
	29	**1024 MB Address Mask** Default Value: 0b Access: R/W Lock FLR Resettable

GMADR_0_2_0_PCI - Graphics Memory Range Address

	This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO) depending on the value of MSAC.APSZ. RO and forced to 0 when MSAC.APSZ >= 1024MB. (i.e. MSAC.APSZ[2]=1)
28	**512MB Address Mask** Default Value: 0b Access: R/W Lock FLR Resettable This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO) depending on the value of MSAC.APSZ. RO and forced to 0 when MSAC.APSZ >= 512MB. (i.e. MSAC.APSZ[1]=1)
27	**256 MB Address Mask** Default Value: 0b Access: R/W Lock FLR Resettable This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO) depending on the value of MSAC.APSZ. RO and forced to 0 when MSAC.APSZ >= 256MB. (i.e. MSAC.APSZ[0]=1)
26:4	**Address Mask** Default Value: 00000000000000000000000b Access: RO Hardwired to 0s to indicate at least 128MB address range.
3	**Prefetchable Memory** Default Value: 1b Access: RO Hardwired to 1 to enable prefetching.
2:1	**Memory Type** Default Value: 10b Access: RO Hardwired to 2h to indicate 64 bit base address.
0	**Memory/IO Space** Default Value: 0b Access: RO Hardwired to 0 to indicate memory space.

Graphics MOCS Register0

GFX_MOCS_0 - Graphics MOCS Register0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0C800h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_0 - Graphics MOCS Register0

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2,1, or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register1

GFX_MOCS_1 - Graphics MOCS Register1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0C804h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_1 - Graphics MOCS Register1

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2,1, or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register2

GFX_MOCS_2 - Graphics MOCS Register2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0C808h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_2 - Graphics MOCS Register2

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2,1, or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register3

GFX_MOCS_3 - Graphics MOCS Register3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0C80Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_3 - Graphics MOCS Register3

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register4

GFX_MOCS_4 - Graphics MOCS Register4

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C810h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_4 - Graphics MOCS Register4

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register5

GFX_MOCS_5 - Graphics MOCS Register5

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C814h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_5 - Graphics MOCS Register5

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register6

GFX_MOCS_6 - Graphics MOCS Register6

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C818h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_6 - Graphics MOCS Register6

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register7

GFX_MOCS_7 - Graphics MOCS Register7

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C81Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_7 - Graphics MOCS Register7

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register8

GFX_MOCS_8 - Graphics MOCS Register8

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C820h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_8 - Graphics MOCS Register8

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register9

GFX_MOCS_9 - Graphics MOCS Register9

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C824h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_9 - Graphics MOCS Register9

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register10

GFX_MOCS_10 - Graphics MOCS Register10

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C828h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_10 - Graphics MOCS Register10

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register11

GFX_MOCS_11 - Graphics MOCS Register11

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C82Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_11 - Graphics MOCS Register11

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register12

GFX_MOCS_12 - Graphics MOCS Register12

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C830h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_12 - Graphics MOCS Register12

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register13

GFX_MOCS_13 - Graphics MOCS Register13

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C834h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_13 - Graphics MOCS Register13

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register14

GFX_MOCS_14 - Graphics MOCS Register14

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C838h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_14 - Graphics MOCS Register14

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register15

GFX_MOCS_15 - Graphics MOCS Register15

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C83Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_15 - Graphics MOCS Register15

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register16

GFX_MOCS_16 - Graphics MOCS Register16

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0C840h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_16 - Graphics MOCS Register16

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register17

GFX_MOCS_17 - Graphics MOCS Register17

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0C844h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_17 - Graphics MOCS Register17

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register18

GFX_MOCS_18 - Graphics MOCS Register18

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0C848h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_18 - Graphics MOCS Register18

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register19

GFX_MOCS_19 - Graphics MOCS Register19

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0C84Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_19 - Graphics MOCS Register19

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register20

GFX_MOCS_20 - Graphics MOCS Register20

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C850h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_20 - Graphics MOCS Register20

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register21

GFX_MOCS_21 - Graphics MOCS Register21

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C854h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_21 - Graphics MOCS Register21

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register22

GFX_MOCS_22 - Graphics MOCS Register22

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A
Size (in bits):	32
Address:	0C858h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_22 - Graphics MOCS Register22

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register23

GFX_MOCS_23 - Graphics MOCS Register23

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C85Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_23 - Graphics MOCS Register23

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register24

GFX_MOCS_24 - Graphics MOCS Register24

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C860h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_24 - Graphics MOCS Register24

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register25

GFX_MOCS_25 - Graphics MOCS Register25

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C864h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_25 - Graphics MOCS Register25

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register26

GFX_MOCS_26 - Graphics MOCS Register26

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C868h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_26 - Graphics MOCS Register26

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register27

GFX_MOCS_27 - Graphics MOCS Register27

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C86Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_27 - Graphics MOCS Register27

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register28

GFX_MOCS_28 - Graphics MOCS Register28

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C870h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_28 - Graphics MOCS Register28

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register29

GFX_MOCS_29 - Graphics MOCS Register29

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C874h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_29 - Graphics MOCS Register29

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register30

GFX_MOCS_30 - Graphics MOCS Register30

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C878h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 00000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_30 - Graphics MOCS Register30

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register31

GFX_MOCS_31 - Graphics MOCS Register31

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C87Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_31 - Graphics MOCS Register31

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register32

GFX_MOCS_32 - Graphics MOCS Register32

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0C880h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_32 - Graphics MOCS Register32

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register33

GFX_MOCS_33 - Graphics MOCS Register33

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0C884h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_33 - Graphics MOCS Register33

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register34

GFX_MOCS_34 - Graphics MOCS Register34

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0C888h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_34 - Graphics MOCS Register34

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register35

GFX_MOCS_35 - Graphics MOCS Register35

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0C88Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 00000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_35 - Graphics MOCS Register35

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register36

GFX_MOCS_36 - Graphics MOCS Register36

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C890h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_36 - Graphics MOCS Register36

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register37

GFX_MOCS_37 - Graphics MOCS Register37

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C894h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_37 - Graphics MOCS Register37

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register38

GFX_MOCS_38 - Graphics MOCS Register38

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C898h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_38 - Graphics MOCS Register38

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register39

GFX_MOCS_39 - Graphics MOCS Register39

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C89Ch

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_39 - Graphics MOCS Register39

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register40

GFX_MOCS_40 - Graphics MOCS Register40

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C8A0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_40 - Graphics MOCS Register40

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register41

GFX_MOCS_41 - Graphics MOCS Register41

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C8A4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_41 - Graphics MOCS Register41

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register42

GFX_MOCS_42 - Graphics MOCS Register42

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C8A8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_42 - Graphics MOCS Register42

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register43

GFX_MOCS_43 - Graphics MOCS Register43

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C8ACh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_43 - Graphics MOCS Register43

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register44

GFX_MOCS_44 - Graphics MOCS Register44

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C8B0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_44 - Graphics MOCS Register44

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register45

GFX_MOCS_45 - Graphics MOCS Register45

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C8B4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_45 - Graphics MOCS Register45

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register46

GFX_MOCS_46 - Graphics MOCS Register46

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C8B8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_46 - Graphics MOCS Register46

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register47

GFX_MOCS_47 - Graphics MOCS Register47

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C8BCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_47 - Graphics MOCS Register47

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register48

GFX_MOCS_48 - Graphics MOCS Register48

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000030 [KBL]
Size (in bits):	32
Address:	0C8C0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_48 - Graphics MOCS Register48

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register49

GFX_MOCS_49 - Graphics MOCS Register49

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000034 [KBL]
Size (in bits):	32
Address:	0C8C4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_49 - Graphics MOCS Register49

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register50

GFX_MOCS_50 - Graphics MOCS Register50

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000038 [KBL]
Size (in bits):	32
Address:	0C8C8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_50 - Graphics MOCS Register50

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 00b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register51

GFX_MOCS_51 - Graphics MOCS Register51

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000031 [KBL]
Size (in bits):	32
Address:	0C8CCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_51 - Graphics MOCS Register51

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 01b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register52

GFX_MOCS_52 - Graphics MOCS Register52

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C8D0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_52 - Graphics MOCS Register52

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register53

GFX_MOCS_53 - Graphics MOCS Register53

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C8D4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_53 - Graphics MOCS Register53

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register54

GFX_MOCS_54 - Graphics MOCS Register54

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C8D8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 000000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_54 - Graphics MOCS Register54

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register55

GFX_MOCS_55 - Graphics MOCS Register55

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C8DCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 00000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_55 - Graphics MOCS Register55

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register56

GFX_MOCS_56 - Graphics MOCS Register56

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C8E0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_56 - Graphics MOCS Register56

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register57

GFX_MOCS_57 - Graphics MOCS Register57

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C8E4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_57 - Graphics MOCS Register57

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register58

GFX_MOCS_58 - Graphics MOCS Register58

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000032 [KBL]
Size (in bits):	32
Address:	0C8E8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_58 - Graphics MOCS Register58

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register59

GFX_MOCS_59 - Graphics MOCS Register59

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000036 [KBL]
Size (in bits):	32
Address:	0C8ECh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_59 - Graphics MOCS Register59

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register60

GFX_MOCS_60 - Graphics MOCS Register60

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003A [KBL]
Size (in bits):	32
Address:	0C8F0h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_60 - Graphics MOCS Register60

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 10b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register61

GFX_MOCS_61 - Graphics MOCS Register61

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000033 [KBL]
Size (in bits):	32
Address:	0C8F4h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_61 - Graphics MOCS Register61

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 00b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register62

GFX_MOCS_62 - Graphics MOCS Register62

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000037 [KBL]
Size (in bits):	32
Address:	0C8F8h

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_62 - Graphics MOCS Register62

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 01b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics MOCS Register63

GFX_MOCS_63 - Graphics MOCS Register63

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000003B [KBL]
Size (in bits):	32
Address:	0C8FCh

MOCS register

DWord	Bit	Description
0	31:15	**Reserved** Default Value: 0000000000000000b Access: RO
	14	**Reserved1** Default Value: 0b Access: RO
	13:11	**Page Faulting Mode** Default Value: 000b Access: R/W This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
	10:8	**Skip Caching control** Default Value: 000b Access: R/W Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is don't care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target
	7	**Enable Skip Caching** Default Value: 0b Access: R/W Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

GFX_MOCS_63 - Graphics MOCS Register63

	6	**Dont allocate on miss** Default Value: 0b Access: R/W Controls defined for RO surfaces in mind, where if the target cache is missed - don't bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
	5:4	**LRU management** Default Value: 11b Access: R/W This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age 3 it tends to stay longer in the cache as compared to older age allocations - 2, 1 or 0. This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Poor chance of generating hits 01: Don't change the LRU if it is a HIT 00: Reserved
	3:2	**Target Cache** Default Value: 10b Access: R/W This field allows the choice of LLC vs eLLC for caching 00: eLLC Only 01: LLC Only 10: LLC/eLLC Allowed 11: LLC/eLLC Allowed
	1:0	**LLC/eDRAM cacheability control** Default Value: 11b Access: R/W Memory type information used in LLC/eDRAM. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB)

Graphics Mode Register

GFX_MODE - Graphics Mode Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1

Address:	0229Ch-0229Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_RCSUNIT

Address:	1229Ch-1229Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VCSUNIT0

Address:	1A29Ch-1A29Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VECSUNIT

Address:	1C29Ch-1C29Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VCSUNIT1

Address:	2229Ch-2229Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_BCSUNIT

This register contains a control bit for the new execlist and 2-level PPGTT functions.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**Execlist Enable** Mask: MMIO#31 When set, software can utilize the execlist registers to load a context into hardware. MI_SET_CONTEXT and MI_ARB_CHECK commands will be converted to NOOP if parsed. When this bit is clear, the Execlist mechanism cannot be used. The context must be loaded via MI_SET_CONTEXT and the ring must be loaded via MMIO access. **Programming Notes**

GFX_MODE - Graphics Mode Register

This bit is *not* intended to be changed dynamically. Changing the value of this bit while rendering is in progress will have UNDEFINED results. This bit should be changed only after a full reset and before submitting *any* commands to the device.

14 **Reserved**

13 **Reserved**

Format:	PBC

12 **Reserved**

11 **Reserved**

10 **Reserved**

Format:	PBC

9 **Per-Process GTT Enable**

Format:	Enable

Per-Process GTT Enable

Value	Name	Description
0h	PPGTT Disable **[Default]**	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.

Programming Notes

This bit is used for enabling PPGTT access in Ring Buffer mode of scheduling. Privilege field in context descriptor states the same in Execlist Mode of scheduling. This field should be set before programming PDP0/1/2/3 registers in order to set the PPGTT translation of memory access.

Programming this bit doesn't enable or disable the PPGTT translation of memory access immediately; the change comes in to affect only when the Page Directory registers are programmed. Programming this bit must be followed by programming Page Directory Registers in order to enable or disable the PPGTT translation of memory access.

8 **Reserved**

8 **Reserved**

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	PBC

7 **64Bit Virtual Addressing Enable**

Format:	Enable

64Bit Virtual Addressing Enable

Value	Name	Description
0h	64Bit Virtual Addressing	When clear indicates GFX operating in 32bit Virtual

GFX_MODE - Graphics Mode Register

	Disable **[Default]**	Addressing for PPGTT based memory access.
1h	64Bit Virtual Addressing Enable	When Set indicates GFX operating in 64bit (48bit Canonical) Virtual Addressing for PPGTT based memory access.

Programming Notes

This bit is only valid when PPGTT is enabled in ring buffer mode of scheduling. Context Descriptor has a similar bit to control 64bit virtual addressing in execlist mode of scheduling. Whether this field is set or clear virtual addresses translated through GGTT are always 32Bit. This field should be programmed before enabling PPGTT access. When this field is not set or for GGTT virtual addresses, Graphics Address [47:32] field of any commands or register exercised by SW should be programmed to 0x0.

6:5 **Reserved**

4 **Reserved**

3 **Reserved**

Format:	MBZ

2:1 **Reserved**

Format:	PBC

0 **Privilege Check Disable**

Format:	Enable

This field when set, disables Privilege Violation checks on non-privileged batch buffers. When set Privileged commands are allowed to be executed from non-privileged batch buffers.

Graphics System Event

GSE_0_2_0_PCI - Graphics System Event	
Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	000E4h

This register can be accessed by either Byte, Word, or Dword PCI config cycles. A write to this register will cause the Graphics System Event display interrupt if it is enabled and unmasked in the display interrupt registers.

DWord	Bit	Description
0	31:24	**GSE Scratch Trigger 3** Default Value: 00000000b Access: R/W
	23:16	**GSE Scratch Trigger 2** Default Value: 00000000b Access: R/W
	15:8	**GSE Scratch Trigger 1** Default Value: 00000000b Access: R/W
	7:0	**GSE Scratch Trigger 0** Default Value: 00000000b Access: R/W

Graphics Translation Table Memory Mapped Range Address

GTTMMADR_0_2_0_PCI - Graphics Translation Table Memory Mapped Range Address

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000004, 0x00000000
Size (in bits):	64
Address:	00010h

This register requests allocation for the combined Graphics Translation Table Modification Range and Memory Mapped Range. The range requires 16 MB combined for MMIO and Global GTT aperture, with 2MB of that used by MMIO, 6MB reserved, and 8MB used by GTT.

GTTADR will begin at (GTTMMADR + 8 MB) while the MMIO base address will be the same as GTTMMADR. The region between (GTTMMADR + 2MB) - (GTTMMADR + 8MB) is reserved.

For the Global GTT, this range is defined as a memory BAR in graphics device config space. It is an alias into which software is required to write Page Table Entry values (PTEs). Software may read PTE values from the global Graphics Translation Table (GTT). PTEs cannot be written directly into the global GTT memory area.

The device snoops writes to this region in order to invalidate any cached translations within the various TLB's implemented on-chip.

The allocation is for 16MB and the base address is defined by bits [38:24].

Note: Per PCI enumeration requirements, to determine the size of a BAR software should write all 1s to the BAR, read it back and see how many of the lower bits read as 0 (meaning that they didn't take the 1s). This indicates the size of the BAR. In order for this to work bits 63 down to the size of the BAR need to be writable to 1s.

DWord	Bit	Description
0	63:39	**Reserved for Memory Base Address** Default Value: 00000000000000000000000b Access: R/W Must be set to 0 since addressing above 512GB is not supported.
	38:24	**Memory Base Address** Default Value: 000000000000000b Access: R/W Set by the OS, these bits correspond to address signals [38:24].
	23:4	**Address Mask** Default Value: 0000000000000000000b Access: RO Hardwired to 0s to indicate at least 16MB address range.

GTTMMADR_0_2_0_PCI - Graphics Translation Table Memory Mapped Range Address

	3	**Prefetchable Memory** Default Value: 0b Access: RO Hardwired to 0 to prevent prefetching.
	2:1	**Memory Type** Default Value: 10b Access: RO Hardwired to 2h to indicate 64 bit base address.
	0	**Memory/IO Space** Default Value: 0b Access: RO Hardwired to 0 to indicate memory space.

GSA_AUDIO_BDF

GSA_AUDIO_BDF

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Access:	R/W
Size (in bits):	32
Address:	1300B0h-1300B3h
Name:	GSA Audio BDF
ShortName:	GSA_AUDIO_BDF
Power:	PG0
Reset:	global

BIOS must program this register with the PCI Bus, Device, and Function of the PCH audio device and set the lock.
Access is a variant of RW-L.
When Lock is not set (bit 0 = 0) the register can be written by any source.
After lock is set (bit 0 = 1), only writes from firmware (cfgspace 0x11111 and srcID 0x10) will update the register values. Writes from other sources will complete without updating the register values.
Any source can read the register.
This register is not reset by the device 2 FLR.

DWord	Bit	Description
0	31:24	**Bus** Default Value: 00000000b Access is RW-L. This field specifies the PCI bus number of the PCH audio device.
	23:19	**Device** Default Value: 00010b Access is RW-L. This field specifies the PCI device number of the PCH audio device.
	18:16	**Function** Default Value: 000b Access is RW-L. This field specifies the PCI function number of the PCH audio device.
	15:1	**Reserved** Format: MBZ
	0	**Lock** Access is RW-KL. This field locks all writeable settings in this register, including itself. Value: 0b — Name: Unlock Value: 1b — Name: Lock

GSA_TOUCH_BDF

GSA_TOUCH_BDF

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Access:	R/W
Size (in bits):	32
Address:	1300B4h-1300B7h
Name:	GSA Touch BDF
ShortName:	GSA_TOUCH_BDF
Power:	PG0
Reset:	global

BIOS must program this register with the PCI Bus, Device, and Function that the Display Engine should use for communication with the PCH touch device, and set the lock.
Access is a variant of RW-L.
When Lock is not set (bit 0 = 0) the register can be written by any source.
After lock is set (bit 0 = 1), only writes from firmware (cfgspace 0x11111 and srcID 0x10) will update the register values. Writes from other sources will complete without updating the register values.
Any source can read the register.
This register is not reset by the device 2 FLR.

DWord	Bit	Description
0	31:24	**Bus** Default Value: 00000000b Access is RW-L. This field specifies the PCI bus number.
	23:19	**Device** Default Value: 00010b Access is RW-L. This field specifies the PCI device number.
	18:16	**Function** Default Value: 000b Access is RW-L. This field specifies the PCI function number.
	15:1	**Reserved** Format: MBZ
	0	**Lock** Access is RW-KL. This field locks all writeable settings in this register, including itself. 0b: Unlock 1b: Lock

GS Invocation Counter

GS_INVOCATION_COUNT - GS Invocation Counter	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02328h

This register stores the number of objects that are part of geometry shader threads. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**GS Invocation Count UDW** Number of objects that are dispatched as a geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)
	31:0	**GS Invocation Count LDW** Number of objects that are dispatched as a geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)

GS Primitives Counter

GS_PRIMITIVES_COUNT - GS Primitives Counter		
Register Space:	MMIO: 0/2/0	
Source:	RenderCS	
Default Value:	0x00000000, 0x00000000	
Access:	R/W	
Size (in bits):	64	
Trusted Type:	1	
Address:	02330h	
This register reflects the total number of primitives that have been output by the Geometry Shader stage. This register is part of the context save and restore..		
DWord	**Bit**	**Description**
0	63:32	**GS Primitives Count UDW** Total number of primitives output by the geometry stage. Updated only when Statistics Enable is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)
	31:0	**GS Primitives Count LDW** Total number of primitives output by the geometry stage. Updated only when Statistics Enable is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)

GT4 Mode Control Register

GT4MODECTL - GT4 Mode Control Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	09038h	
GT4 Mode Control Register		
DWord	**Bit**	**Description**
0	31:18	**RSVD** Access: R/W
	17:10	**Reserved**
	9:2	**Reserved**
	1:0	**GT4 Mode Control** Access: R/W GT4 Usage mode: 00b: Non-GT4. 01b: GT4 is used in Alternate Frame rendering Mode (AFR). 10b: Basic Split Frame rendering Mode (SFR). 11b: Complex Split Frame rendering Mode (SFR w/ CBR).

GTC_CTL

GTC_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	67000h-67003h
Name:	Global Time Code Control
ShortName:	GTC_CTL
Power:	PG1
Reset:	soft

DWord	Bit	Description
0	31	**GTC Function Enable** This bit enables the GTC counter. Value / Name: 0b – Disable 1b – Enable Restriction: Restriction : Enable this bit before enabling GTC controller operation on a port with a GTC capable device.
	30:29	**Reserved** Format: MBZ
	28:13	**Reserved**
	12:1	**Reserved** Format: MBZ
	0	**Reserved**

GTC_DDA_M

GTC_DDA_M		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	67010h-67013h	
Name:	Global Time Code DDA M	
ShortName:	GTC_DDA_M	
Power:	PG1	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:24	**Reserved**
	23:0	**GTC DDA M** This field is used to program the M value of the GTC DDA. The ratio of M to N programmed depends on the GTC reference clock. The DDA programmed values are related by the following formula: 1/(accumulator increment) = Reference Clock * DDA_M / DDA_N

GTC_DDA_N

GTC_DDA_N		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	67014h-67017h	
Name:	Global Time Code DDA N	
ShortName:	GTC_DDA_N	
Power:	PG1	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:24	**GTC Accum Inc** Format: U7.1 This field is the GTC accumulator increment value in nanoseconds each time the DDA trips. It is programmed in 7.1 fixed point binary format where the LSB represents 0.5ns increment.
	23:0	**GTC DDA N** This field is used to program the N value of the GTC DDA. The ratio of M to N programmed depends on the GTC reference clock and should not result in any accumulation error in any 10ms interval period. The DDA programmed values are related by the following formula: 1/(accumulator increment) = Reference Clock * DDA_M / DDA_N

GTC_IIR

GTC_IIR	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/WC
Size (in bits):	32
Address:	67058h-6705Bh
Name:	Global Time Code Interrupt Identity
ShortName:	GTC_IIR
Power:	PG1
Reset:	soft

See the GTC interrupt bit definition to find the source event for each interrupt bit.

DWord	Bit	Description
0	31:0	**Interrupt Identity Bits** This field holds the persistent values of the GTC interrupt bits which are unmasked by the GTC_IMR. Bits set in this register will propagate to the GTC interrupt in the Display Engine Miscellaneous Interrupts. Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits.

Value	Name
0b	Condition Not Detected
1b	Condition Detected

GTC_IMR

GTC_IMR	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	67054h-67057h
Name:	Global Time Code Interrupt Mask
ShortName:	GTC_IMR
Power:	PG1
Reset:	soft

See the GTC interrupt bit definition to find the source event for each interrupt bit.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** This field contains a bit mask which selects which GTC events are reported int the GTC IIR.

Value	Name
0b	Not Masked
1b	Masked
FFFFFFFFh	All interrupts masked **[Default]**

GTC_LIVE

GTC_LIVE		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Address:	67020h-67023h	
Name:	Global Time Code Live	
ShortName:	GTC_LIVE	
Power:	PG1	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GTC Live Value** This field contains the live current value of the GTC. It is inactive when the GTC controller function is disabled. This register also samples and holds the live GTC value following a Audio Time Capture (ATC) event until software reads this register. A subsequent read of this register will reflect the live value.

GTC_PORT_CTL

GTC_PORT_CTL	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	64070h-64073h
Name:	DDI A GTC Port Control
ShortName:	GTC_PORT_CTL_A
Power:	PG1
Reset:	soft
Address:	64170h-64173h
Name:	DDI B GTC Port Control
ShortName:	GTC_PORT_CTL_B
Power:	PG2
Reset:	soft
Address:	64270h-64273h
Name:	DDI C GTC Port Control
ShortName:	GTC_PORT_CTL_C
Power:	PG2
Reset:	soft
Address:	64370h-64373h
Name:	DDI D GTC Port Control
ShortName:	GTC_PORT_CTL_D
Power:	PG2
Reset:	soft

DWord	Bit	Description
0	31	**Port Global Time Code Enable** This bit enables the GTC controller to start lock acquisition phase with remote GTC sink connected to this port. This bit has no effect if the GTC controller is disabled. **Value** / **Name** 0b – Disable 1b – Enable **Restriction**

GTC_PORT_CTL

		Restriction : The Maintenance Phase Enable bit must be initially written as '0' when this bit is set.
	30:25	**Reserved**
	24	**Maintenance Phase Enable** This bit is used by software to transition from lock acquisition to lock maintenance phase. The GTC controller generates an interrupt at the end of the lock phase as determined by lock acquisition duration field. Software shall read the sink device GTC lock done bit. If set, software shall set this bit to '1' after first writing the GTC skew value to the RX GTC skew DPCD offset with GTC skew enable bit set to '1'.

Value	Name	Description
0b	Lock	Lock acquisition phase. The controller writes or reads GTC every 1ms.
1b	Maintain	Lock maintenance phase. The controller writes or reads GTC every 10ms.

	23:1	**Reserved**
	0	**Port RX Lock Done** This bit indicates the remote GTC sink has achieved lock. This bit shall be written by software after reading remote GTC sink DPCD register. This bit shall be cleared by software when GTC controller is reset from lock maintenance mode to lock acquisition mode or when the controller is disabled.

Value	Name
0b	Not Locked
1b	Locked

GTC_PORT_MISC

GTC_PORT_MISC		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00034A00	
Access:	R/W	
Size (in bits):	32	
Address:	64094h-64097h	
Name:	DDI A GTC Port Miscellaneous	
ShortName:	GTC_PORT_MISC_A	
Power:	PG1	
Reset:	soft	
Address:	64194h-64197h	
Name:	DDI B GTC Port Miscellaneous	
ShortName:	GTC_PORT_MISC_B	
Power:	PG2	
Reset:	soft	
Address:	64294h-64297h	
Name:	DDI C GTC Port Miscellaneous	
ShortName:	GTC_PORT_MISC_C	
Power:	PG2	
Reset:	soft	
Address:	64394h-64397h	
Name:	DDI D GTC Port Miscellaneous	
ShortName:	GTC_PORT_MISC_D	
Power:	PG2	
Reset:	soft	

DWord	Bit	Description
0	31:22	**Reserved** Format: MBZ
	21:12	**GTC Update Message Delay** Default Value: 00110100b 52 nanoseconds This field programs the absolute delay in nanoseconds between the GTC at the aux sync point event and the corresponding GTC value at the capture point. It represents the delay between the GTC values at the aux sync point and capture point introduced due to synchronization and glitch suppression.

GTC_PORT_MISC		
	11:8	**Min Lock Duration** Default Value: 1010b 10ms This field determines the minimum duration in milliseconds of lock acquisition and maintenance phase after which software is notified through interrupt. The GTC interrupt enable and mask register must be enabled beforehand. Software may also poll the interrupt identity bit in IIR.
	7:0	**Reserved** Format: MBZ

GTC_PORT_TX_CURR

GTC_PORT_TX_CURR	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	64078h-6407Bh
Name:	DDI A GTC Port TX Current
ShortName:	GTC_PORT_TX_CURR_A
Power:	PG1
Reset:	soft
Address:	64178h-6417Bh
Name:	DDI B GTC Port TX Current
ShortName:	GTC_PORT_TX_CURR_B
Power:	PG2
Reset:	soft
Address:	64278h-6427Bh
Name:	DDI C GTC Port TX Current
ShortName:	GTC_PORT_TX_CURR_C
Power:	PG2
Reset:	soft
Address:	64378h-6437Bh
Name:	DDI D GTC Port TX Current
ShortName:	GTC_PORT_TX_CURR_D
Power:	PG2
Reset:	soft

Description
There is one instance of this register per port A, B, C, and D.

DWord	Bit	Description
0	31:0	**Global Time Code Port TX Current** This field contains the local GTC value sampled at the Aux sync point of the response message from the remote GTC sink following software read of the remote sink GTC DPCD register.

GTC_PORT_TX_PREV

GTC_PORT_TX_PREV	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	64080h-64083h
Name:	DDI A GTC Port TX Previous
ShortName:	GTC_PORT_TX_PREV_A
Power:	PG1
Reset:	soft
Address:	64180h-64183h
Name:	DDI B GTC Port TX Previous
ShortName:	GTC_PORT_TX_PREV_B
Power:	PG2
Reset:	soft
Address:	64280h-64283h
Name:	DDI C GTC Port TX Previous
ShortName:	GTC_PORT_TX_PREV_C
Power:	PG2
Reset:	soft
Address:	64380h-64383h
Name:	DDI D GTC Port TX Previous
ShortName:	GTC_PORT_TX_PREV_D
Power:	PG2
Reset:	soft

Description
There is one instance of this register per port A, B, C, D and F.

DWord	Bit	Description
0	31:0	**Global Time Code Port TX Previous** This field contains the previous local GTC value sampled at Aux sync point. It is transferred from the GTC_PORT_TX_CURR register when the current value is updated.

GTFORCEAWAKE

GTFORCEAWAKE	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	130090h-130093h
Name:	GT Force Awake
ShortName:	GTFORCEAWAKE
Power:	PG0
Reset:	soft

DWord	Bit	Description
0	31:1	**Reserved**
	0	**Force Awake** This field is no longer used. The multiple force wake mechanism has replaced it. Refer to MULTIFORCEWAKE 0xA188 register description for the usage.

GT Function Level Reset Control Message

FLRCTLMSG - GT Function Level Reset Control Message		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	08100h	
GT FLR Control Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:1	**Reserved** Access: RO Reserved
	0	**Initiate GT Function Level Reset Message** Access: R/W Set GT Function Level Reset (FLR) 1: Initiate GT FLR - This is a Non-Posted message to reset Render, Media, Blitter and GTI-Device domains. - This bit is cleared by the CPunit upon completion of the reset.

GT Interrupt 0 Definition

GT Interrupt 0 Definition	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44300h-4430Fh
Name:	GT 0 Interrupts
ShortName:	GT_0_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers.
Bits 15:0 are used for Render CS.
Bits 31:16 are used for Blitter CS.
The IER enabled Render Interrupt IIR (sticky) bits are ORed together to generate the Render Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled Blitter Interrupt IIR (sticky) bits are ORed together to generate the Blitter Interrupts Pending bit in the Master Interrupt Control register.
0x44300 = ISR
0x44304 = IMR
0x44308 = IIR
0x4430C = IER

DWord	Bit	Description
0	31	**Spare 31**
	30	**Spare 30**
	29	**Spare 29**
	28	**Spare 28**
	27	**BCS Wait On Semaphore**
	26	**Spare 26**
	25	**Spare 25**
	24	**BCS Context Switch Interrupt**
	23	**Spare 23**
	22	**Spare 22**
	21	**Spare 21**
	20	**BCS MI Flush DW Notify**
	19	**BCS Error Interrupt**
	18	**Spare 18**

GT Interrupt 0 Definition		
	17	**Spare 17**
	16	**BCS MI User Interrupt**
	15	**Spare 15**
	14	**EU Restart Interrupt**
	13	**Spare 13**
	12	**Spare 12**
	11	**CS Wait On Semaphore**
	10	**CS L3 Counter Save**
	9	**CS TR Invalid Tile Detection**
	8	**CS Context Switch Interrupt**
	7	**Page Fault Interrupt** This interrupt is for handling Legacy Page Fault interface for all Command Streamers [BCS, RCS, VCS, VECS]. When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c "page fault support" section for more details.
	6	**CS Watchdog Counter Expired**
	5	**Spare 5**
	4	**CS PIPE_CONTROL Notify**
	3	**CS Error Interrupt**
	2	**Spare 2**
	1	**Reserved**
	0	**CS MI User Interrupt**

GT Interrupt 1 Definition

GT Interrupt 1 Definition	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44310h-4431Fh
Name:	GT 1 Interrupts
ShortName:	GT_1_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the GT Interrupt 1 registers.
Bits 15:0 are used for VCS1.
Bits 31:16 are used for VCS2.
The VCS1 Interrupt IIR (sticky) bits are ORed together to generate the VCS1 Interrupts Pending bit in the Master Interrupt Control register.
The VCS2 Interrupt IIR (sticky) bits are ORed together to generate the VCS2 Interrupts Pending bit in the Master Interrupt Control register.
0x44310 = ISR
0x44314 = IMR
0x44318 = IIR
0x4431C = IER

DWord	Bit	Description
0	31	**Spare 31**
	30	**Spare 30**
	29	**Spare 29**
	28	**Spare 28**
	27	**VCS2 Wait On Semaphore**
	26	**Spare 26**
	25	**Reserved**
	24	**VCS2 Context Switch Interrupt**
	23	**Spare 23**
	22	**VCS2 Watchdog Counter Expired**
	21	**Reserved**
	20	**VCS2 MI Flush DW Notify**
	19	**VCS2 Error Interrupt**
	18	**Spare 18**

GT Interrupt 1 Definition		
	17	**Spare 17**
	16	**VCS2 MI User Interrupt**
	15	**Spare 15**
	14	**Spare 14**
	13	**Spare 13**
	12	**Spare 12**
	11	**VCS1 Wait On Semaphore**
	10	**Spare 10**
	9	**Reserved**
	8	**VCS1 Context Switch Interrupt**
	7	**Spare 7**
	6	**VCS1 Watchdog Counter Expired**
	5	**Reserved**
	4	**VCS1 MI Flush DW Notify**
	3	**VCS1 Error Interrupt**
	2	**Spare 2**
	1	**Spare 1**
	0	**VCS1 MI User Interrupt**

GT Interrupt 2 Definition

GT Interrupt 2 Definition	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44320h-4432Fh
Name:	GT 2 Interrupts
ShortName:	GT_2_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the GT Interrupt 2 registers.
Bits 15:0 are used for GTPM.
The IER enabled GTPM Interrupt IIR (sticky) bits are ORed together to generate the GTPM Interrupts Pending bit in the Master Interrupt Control register.
0x44320 = ISR
0x44324 = IMR
0x44328 = IIR
0x4432C = IER

DWord	Bit	Description
0	31	**Reserved**
	30	**Reserved**
	29	**Reserved**
	28	**Reserved**
	27	**Reserved**
	26	**Reserved**
	25	**Reserved**
	24	**Reserved**
	23	**Reserved**
	22	**Reserved**
	21	**Reserved**
	20	**Reserved**
	19	**Reserved**
	18	**Reserved**
	17	**Reserved**
	16	**Reserved**
	15	**Spare 15**

GT Interrupt 2 Definition		
	14	**Spare 14**
	13	**Unslice Frequency Control Up Interrupt**
	12	**Unslice Frequency Control Down Interrupt**
	11	**NFADFL Frequency Up Interrupt**
	10	**NFADFL Frequency Down Interrupt**
	9	**Reserved**
	8	**GTPM Engines Idle Interrupt**
	7	**GTPM Uncore to Core Trap Interrupt**
	6	**GTPM Render Frequency Downwards Timeout During RC6 Interrupt**
	5	**GTPM Render P-State Up Threshold Interrupt**
	4	**GTPM Render P-State Down Threshold Interrupt**
	3	**Spare 3**
	2	**GTPM Render Geyserville Up Evaluation Interval Interrupt**
	1	**GTPM Render Geyserville Down Evaluation Interval Interrupt**
	0	**Spare 0**

GT Interrupt 3 Definition

GT Interrupt 3 Definition

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	44330h-4433Fh
Name:	GT 3 Interrupts
ShortName:	GT_3_INTERRUPT
Power:	PG0
Reset:	soft

This table indicates which events are mapped to each bit of the GT Interrupt 3 registers.
Bits 15:0 are used for VEBox.
Bits 27:16 are Reserved.
Bits 31:28 are used for OACS.
The VEBox Interrupt IIR (sticky) bits are ORed together to generate the VEBox Interrupts Pending bit in the Master Interrupt Control register.
0x44330 = ISR
0x44334 = IMR
0x44338 = IIR
0x4433C = IER

DWord	Bit	Description
0	31	**Spare 31**
	30	**Spare 30**
	29	**Spare 29**
	28	**Performance Monitoring Buffer Half-Full Interrupt** For internal trigger (timer event based) reporting, this interrupt is generated if the report buffer crosses the half full limit.
	27	**Spare 27**
	26	**Spare 26**
	25	**Spare 25**
	24	**Spare 24**
	23	**Spare 23**
	22	**Spare 22**
	21	**Spare 21**
	20	**Spare 20**
	19	**Spare 19**
	18	**Spare 18**

GT Interrupt 3 Definition		
	17	**Reserved**
	16	**Reserved**
	15	**Spare 15**
	14	**Spare 14**
	13	**Spare 13**
	12	**Spare 12**
	11	**VECS Wait On Semaphore**
	10	**Spare 10**
	9	**Spare 9**
	8	**VECS Context Switch Interrupt**
	7	**Spare 7**
	6	**Reserved**
	5	**Spare 5**
	4	**VECS MI Flush DW Notify**
	3	**VECS Error Interrupt**
	2	**Spare 2**
	1	**Spare 1**
	0	**VECS MI User Interrupt**

GTI PFET control register with lock

PFETCTL - GTI PFET control register with lock

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0004005A [KBL]
Size (in bits):	32
Address:	24008h

DWord	Bit	Description
0	31	**PFET Control Lock** Access: R/W Lock 0 = Bits of GTI PFETCTL register are R/W 1 = All bits of GTI PFETCTL register are RO (including this lock bit) Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	30:23	**Reserved** Access: RO Reserved
	22	**Leave firewall disabled** Access: R/W Lock When this bit is set SPC will not firewall the gated domain for a power down flow. But it will pretend to complete the flow with PM. Encodings: 0 = Default mode, that is, firewall gated domain to ungated domain crossing during power down flows 1 = Leave firewall disabled, that is, don't firewall the gated domain, but complete logical flow
	21	**Leave FET On** Access: R/W Lock When This bit is set SPC will not turn off the PFET eventhough it will complete the flow with PM Encodings: 0 = Default mode, that is, power off fets during power down flows 1 = Leave ON mode, that is, don't power off pfet, but complete logical flow **Programming Notes** This bit should be programmed before the powerup sequence is initiated for GTI.
	20	**Power Well Status** Access: R/WC 0 = Well is powered Down 1 = Well is powered Up Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.

PFETCTL - GTI PFET control register with lock

	19	**Powergood timer error** Access: R/WC 0 = Well is powered Down 1 = Well is powered Up Once written to 1, the lock is set and cannot be cleared (that is, writing a 0 will not clear the lock). These bits are not reset on FLR.
	18:16	**Delay from enabling secondary PFETs to power good** Access: R/W Lock Delay from enabling secondary PFETs to power good 3'b000: SKL - 40ns 3'b001: SKL - 80ns, 3'b010: SKL - 160ns 3'b011: SKL - 320ns, 3'b100: SKL - 640ns 3'b101: SKL - 1280ns 3'b110: SKL - 2560ns 3'b111: SKL - 5120ns **Value** / **Name**: 100b / **[Default]**
	15:13	**Time period last primay pfet strobe to secondary pfet strobe** Access: R/W Lock Time period last primary pfet strobe to secondary pfet strobe 3'b000: SKL - 10ns (or 1 bclk) 3'b001: SKL - 20ns (or 2 bclk)) 3'b010: SKL - 30ns (or 3 bclk) 3'b111: SKL - 80ns (or 8 bclk)
	12:10	**Time period b/w two adjacent strobes** Access: R/W Lock Time period b/w two adjacent strobes to the primary FETs 3'b000: SKL - 10ns (or 1 bclk) 3'b001: SKL - 20ns (or 2 bclk) 3'b010: SKL - 30ns (or 3 bclk) 3'b111: SKL - 80ns (or 8 bclk)
	9:7	**FET setup margin from enable to strobe** Access: R/W Lock Setup margin in design before sampling enable event at the first pre-charge sequencer/shift register flop 3'b000: SKL - 10ns (or 1 bclk) 3'b001: SKL - 20ns (or 2 bclk) 3'b010: SKL - 30ns (or 3 bclk) 3'b111: SKL - 80ns (or 8 bclk)

PFETCTL - GTI PFET control register with lock

	6:0	**Number of flops to enable primary FETs** Access: R/W Lock Number of flops to enable primary FETs. For a setting of N there will be N+1 total strobes generated 7'b0000000: 10 Flops to be strobed 7'b0000001: 11 Flops to be strobed 7'b0000010: 12 Flops to be strobed 7'b0001111: 26 Flops to be strobed

Value	Name
1011010b	**[Default]**

GTI Power Gate Control Request

PGCTLREQ - GTI Power Gate Control Request		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	24000h	
Clock Gating Messages Register		
DWord	**Bit**	**Description**
0	31:16	**Message Mask** Access: RO Message Mask - To write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:1	**Reserved** Access: RO Reserved
	0	**Power Gate Request** Access: R/W Media1 power well request: '0' : Initiate Power Down request '1' : Initiate Power UP req

GT Mode Register

GT_MODE - GT Mode Register

Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000 [KBL]
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	07008h

This Register is used to control the 6EU and 12EU configuration for GT.
Writing 0x01FF01FF to this register enables the 6EU mode.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**EU Local Thread Checking Enable** Access: r/w This field configures the EU local thread checking. If enable the stateless access will be checked against the local thread's scratch space size and start address. 0h – Disable **[Default]** – EU local thread checking is disabled. 1h – Enable – EU local thread checking is enabled.
	14:13	**SFR mode** Access: r/w Format: U2 This field must be zero when not in GT4(SFR) configuration i.e GTB_rendermode fuse set to SFR.
	12:11	**Cross Slice Hashing Mode** Access: r/w Format: U2

Value	Name	Description	Programming Notes
0h	Normal Mode **[Default]**	GT3: 16x16 Hashing enabled GT2 or lower modes: No cross slice hashing	

GT_MODE - GT Mode Register

1h	Cross Slice Hashing Disable	Disables the cross slice hashing	
2h	32x16 Hahsing	32x16 Pixel hashing across slices	
3h	32X32 hashing	32X32 pixel hashing across slices	This setting must be used when sub-slice hashing mode is 16x16.

Programming Notes
Normal mode of operation in GT3 mode will be to use either 16x16 Hashing or 32x32 Hashing.

10 **Reserved**

Access:	r/w
Format:	PBC

9:8 **Subslice Hashing Mode**

Access:	r/w
Format:	U2

This field defines hashing modes across subslices.

Value	Name	Description
0h	**[Default]**	8X8 hashing
1h		16x4 hashing
2h		8x4 hashing
3h		16x16 hashing

7 **Reserved**

Access:	r/w
Format:	PBC

7 **Reserved**

Access:	r/w
Format:	PBC

7 **Bindless Surface Base Address Select**

Access:	r/w
Format:	Enable

This field selects Bindless_Surface_State_Base_Addr versus Dynamic_State_Base_Addr for sampler state heap base address. It only applies when Bindless Surfaces are being enabled via the BTI encoding.

Value	Name	Description
0h	Disable **[Default]**	Selects Dynanmic State Base Addr for Sampler State.
1h	Enable	Selects Bindless Surface Base Address for Sampler State

GT_MODE - GT Mode Register

	6	**Reserved** Access: r/w Format: PBC
	5:4	**Slice2 IZ Hashing: 7 EU subslice encoding** Access: r/w These bits control 3-way sub-slice hashing by conveying which sub-slice has 7 EUs.

Value	Name	Description
0h	**[Default]**	All subslices have equal number of EUs.
1h		Subslice 2 has 7 EUs.
2h		Subslice 1 has 7 EU.
3h		Subslice 0 has 7 EUs.

Programming Notes
SW must program these bits based on EU Disable Fuses in Slice 2.

	3:2	**Slice1 IZ Hashing: 7 EU subslice encoding** Access: r/w These bits control 3-way sub-slice hashing by conveying which sub-slice has 7 EUs.

Value	Name	Description
0h	**[Default]**	All subslices have equal number of EUs.
1h		Subslice 2 has 7 EUs.
2h		Subslice 1 has 7 EUs.
3h		Subslice 0 has 7 EUs.

Programming Notes
SW must program these bits based on EU Disable Fuses in Slice 1.

	1:0	**Slice 0 IZ Hashing: 7 EU subslice encoding** Access: r/w These bits control 3-way sub-slice hashing by conveying which sub-slice has 7 EUs.

Value	Name	Description
0h	**[Default]**	All subslices have equal number of EUs.
1h		Subslice 2 has 7 EUs.
2h		Subslice 1 has 7 EUs.
3h		Subslice 0 has 7 EUs.

Programming Notes
SW must program these bits based on EU Disable Fuses in Slice 0.

GTSCRATCH

GTSCRATCH	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	4F100h-4F11Fh
Name:	GT Scratchpad
ShortName:	GTSCRATCH_*
Power:	PG0
Reset:	soft

There are 8 instances of this register format.

Restriction
Restriction : These registers are used by hardware and must not be used by software.

DWord	Bit	Description
0	31:0	**GT Sratchpad** GT Scratchpad

GTSP0

GTSP0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130040h-130043h	
Name:	GT Scratchpad 0	
ShortName:	GTSP0	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP1

GTSP1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130044h-130047h	
Name:	GT Scratchpad 1	
ShortName:	GTSP1	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:16	**GT scratch pad**
	15:0	**Multiple Force Wake** GT programs this field with the multiple force wake status. Software reads this field to find the status. Refer to MULTIFORCEWAKE 0xA188 register description for the usage.

GTSP2

GTSP2		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130048h-13004Bh	
Name:	GT Scratchpad 2	
ShortName:	GTSP2	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP3

GTSP3		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	13004Ch-13004Fh	
Name:	GT Scratchpad 3	
ShortName:	GTSP3	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP4

GTSP4		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130050h-130053h	
Name:	GT Scratchpad 4	
ShortName:	GTSP4	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP5

GTSP5		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130054h-130057h	
Name:	GT Scratchpad 5	
ShortName:	GTSP5	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP6

GTSP6		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	130058h-13005Bh	
Name:	GT Scratchpad 6	
ShortName:	GTSP6	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:0	**GT scratch pad**

GTSP7

GTSP7		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Access:	R/W	
Size (in bits):	32	
Address:	13005Ch-13005Fh	
Name:	GT Scratchpad 7	
ShortName:	GTSP7	
Power:	PG0	
Reset:	soft	
DWord	**Bit**	**Description**
0	31:1	**GT scratch pad**
	0	**Reserved**

GTT Cache Enable

GTT_CACHE_EN - GTT Cache Enable

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000 [KBL]
Size (in bits):	32
Address:	04024h

Enable GTT Cache for respective client(s), A0: Must program/observed this to all 0 due to Big Pages Bug 1898112

DWord	Bit	Description
0	31:0	**GTT Cache Enable for CS** Access: R/W Enable GTT Caching for all the client(s) below: 31: BLIT Engine (overrides individual enables of the units) 30: VEBX Engine (overrides individual enables of the units) 29: MFX Engine (overrides individual enables of the units) 28: GFX Engine (overrides individual enables of the units) 27-15: Reserved 14: VMCunit 13: VLFunit 12: BLBunit 11: VFWunit 10: VEOunit 9: HIZunit 8: RCZunit 7: RCCunit 6: ISCunit 5: DCunit 4: MTunit 3: SOLunit 2: VFunit 1: RSunit 0: CSunit Value: 00000000h — Name: **[Default]**

Hardware Scratch Read Write

HSRW_0_2_0_PCI - Hardware Scratch Read Write

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	16
Address:	00060h

This register is reserved as a HW scratchpad.

DWord	Bit	Description
0	15:0	**Reserved R/W** Default Value: 0000000000000000b Access: R/W Reserved for future usage.

Hardware Status Mask Register

HWSTAM - Hardware Status Mask Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02098h-0209Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_RCSUNIT
Address:	12098h-1209Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VCSUNIT0
Address:	1A098h-1A09Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VECSUNIT
Address:	1C098h-1C09Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VCSUNIT1
Address:	22098h-2209Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_BCSUNIT

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are mask bits that prevent the corresponding bits in the Interrupt Status Register from generating a Hardware Status Write (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

DWord	Bit	Description
0	31:0	**Hardware Status Mask** Format: Array of Masks Refer to the Interrupt Control Register section for bit definitions. Reserved bits are RO.

HWSTAM - Hardware Status Mask Register

Value	Name
FFFFFFFFh	**[Default]**

Hardware Status Page Address Register

HWS_PGA - Hardware Status Page Address Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Address:	02080h-02083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_RCSUNIT

Address:	12080h-12083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VCSUNIT0

Address:	1A080h-1A083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VECSUNIT

Address:	1C080h-1C083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VCSUNIT1

Address:	22080h-22083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_BCSUNIT

This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory.

DWord	Bit	Description
0	31:12	**Address** Format: GraphicsAddress[31:12] This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the Hardware Status Page. The Global GTT is used to map this page from the graphics virtual address to physical address. **Programming Notes** If the Per-Process Virtual Address Space and Exec List Enable bit is set, HW requires that the status page is programmed to allow for the context switch status to be reported.
	11:0	**Reserved** Format: MBZ

HCP Bitstream Output Minimal Size Padding Count Report Register

HCP_MINSIZE_PADDING_COUNT - HCP Bitstream Output Minimal Size Padding Count Report Register

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E9B4h

This register stores the count in bytes of **minimal size padding insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**HCP MinSize Padding Count** Format: U32 Total number of bytes in the bitstream output contributing to minimal size padding operation. This count is updated each time when the padding count is incremented.

HCP CABAC Status

HCP_CABAC_STATUS - HCP CABAC Status

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Size (in bits):	32
Trusted Type:	1
Address:	1E904h

HCP CABAC status.

DWord	Bit	Description
0	31:12	**Reserved** Format: MBZ
	11	**Temporal Direction Motion Vector Out-of-Bound Error** Default Value: 0 Access: RO Format: U1 This flag indicates motion vectors calculated from the Temporal Direct Motion vector is larger than the allowed range.
	10:7	**Reserved** Format: MBZ
	6	**Motion Vector Delta SE** Default Value: 0 Access: RO Format: U1 This flag indicates out-of-bound motion vector delta SEs coded in the bit-stream.
	5	**Delta QP SE** Default Value: 0 Access: RO Format: U1 This flag indicates leading-one overflow during CABAC decode of cu_qp_delta_abs.
	4	**Residual Error** Default Value: 0 Access: RO Format: U1 This flag indicates out-of-bound absolute coefficient level SEs coded in the bit-stream.

HCP_CABAC_STATUS - HCP CABAC Status

	3	**Slice and Error** Default Value: 0 Access: RO Format: U1 This flag indicates a pre-mature end to the slice or an inconsistent end of slice on the last Ctb of a slice.
	2:1	**Reserved** Format: MBZ
	0	**Ctb Concealment Flag** Default Value: 0 Access: RO Format: U1 Each pulse from this flag indicates one Ctb is concealed by the HCP.

HCP Decode Status

HCP_DEC_STATUS - HCP Decode Status

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E900h

HCP Decode status.

DWord	Bit	Description
0	31:18	**Number of Ctbs Concealed** Default Value: 0 Format: U14 This 16-bit field indicates the number of Ctbs concealed during the decoding of the current frame. This field is cleared with the HCP_PIPE_MODE_SELECT command.
	17	**Frame Dec Active** Default Value: 0 Format: U1 This flag indicates that the decoder hardware is actively decoding a picture.
	16	**Indirect Bitstream ObjectAccess Upper Bound Error** Default Value: 0 Format: U1 This flag indicates that the upper bound bit-stream address was reached.
	15:0	**Bit-stream Error Flags** Default Value: 0 Format: U16 This 16-bit field indicates the number of bit stream errors detected for each bit field indicated in the CABAC Status register.

HCP Frame BitStream BIN Count

HCP_BIN_CT - HCP Frame BitStream BIN Count

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x723BA5C0
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E980h

This register stores the number of BINs decoded in a frame.
This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**HCP Frame Bit-stream BIN Count** Default Value: 723ba5c0h Format: U32 Total number of BINs decoded/ in current frame. This number is used with frame performance count to derive Bin/clk.

HCP Frame Motion Comp Miss Count

HCP_MISS_CT - HCP Frame Motion Comp Miss Count

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E988h

This register stores the total number of cacheline hits occurred in the motion compensation cache per frame.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**HCP Frame Motion Comp cache miss Count** Format: U16 Total number of CL misses occurred in the 12KB cache of the motion compensation engine per frame. This number is used along with **HCP Frame Motion Comp Read Count** to derive motion comp cache miss/hit ratio.

HCP Frame Motion Comp Read Count

HCP_READ_CT - HCP Frame Motion Comp Read Count

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E984h

This register stores the total number of reference picture read requests made by the Motion Compensation engine per frame.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ
	19:0	**HCP Frame Motion Comp CL read request Count** Format: U20 Total number of reference picture read requests by the motion compensation engine per frame.

HCP Frame Performance Count

HCP_FRAME_PERF_CNT - HCP Frame Performance Count

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E960h

This register stores the number of clock cycles spent decoding/encoding the current frame.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**Count** Format: U32 Total number of clocks between frame start and frame end. This count is incremented on crm_clk.

HCP Image Status Control

HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	1E9BCh

DWord	Bit	Description
0	31:24	**Cumulative Frame Delta QP/QIndex** Format: S7 Used for Frame Level Multi-pass Rate Control. HEVC: cu_qp = input (first pass) cu_qp + Cumulative Frame Delta Qp. Pak does clamping to max value based on bitdepth. Bit31 is the sign bit. VP9: cu_qindex = input (first pass) cu_qindex + Cumulative Frame Delta Qindex. Pak does clamping to -127..127 after adding. Bit31 is the sign bit.
	23	**Reserved** Format: MBZ
	22:16	**Cumulative Frame Delta LF** Access: RO Format: S6 Used for Frame Level Multi-pass Rate Control. LF_level = input (first pass) LF_level + Cumulative Frame Delta LF level. Pak does clamping to -63..63 after adding.
	15:12	**Reserved** Format: MBZ
	11:8	**Total Num-Pass** Format: U4
	7:3	**Reserved** Format: MBZ
	2	**Frame Bit Count Violate - under run** Access: RO Format: U1 This can trigger Frame Level Multi-pass Rate Control. Set to 1 if frame bit count is less than or equal to FrameBitRateMin

HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control

	1	**Frame Bit Count Violate - over run** Access: RO Format: U1 This can trigger Frame Level Multi-pass Rate Control. Set to 1 if frame bit count is less than or equal to FrameBitRateMax
	0	**LCU Bit Count Violate- overrun**

HCP Image Status Mask

HCP_IMAGE_STATUS_MASK - HCP Image Status Mask		
Register Space:	MMIO: 0/2/0	
Source:	VideoCS	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Trusted Type:	1	
Address:	1E9B8h	
This register stores the image status(flags).		
DWord	**Bit**	**Description**
0	31:3	**Reserved** Format: MBZ
	2	**FrameBitRateMinReportMask** Same as **FrameSzUnderStatusEn** in HCP_PIC_STATE.
	1	**FrameBitRateMaxReportMask** Same as **FrameSzOverStatusEn** in HCP_PIC_STATE.
	0	**FrameLcuMaxReportMask**

HCP Memory Latency Count1

HCP_LAT_CT1 - HCP Memory Latency Count1

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E968h

This register stores the max and min memory latency counts reported on reference read requests.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** Format: U8 This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** Format: U8 This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the motion compensation engine is most likely hung waiting for read data to be returned from sub-system.
	15:8	**HCP Reference picture read request - Max Latency Count in 8xMedia clock cycles** Format: U8 This field reports the maximum memory latency count on all reference reads requested by the motion compensation engine.
	7:0	**HCP Reference picture read request - Min Latency Count in 8xMedia clock cycles** Format: U8 This field reports the minimum memory latency count on all reference reads requested by the motion compensation engine.

HCP Memory Latency Count2

HCP_LAT_CT2 - HCP Memory Latency Count2

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E96Ch

This register stores the accumulative memory latency count on reference picture read requests.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	25:0	**HCP Reference picture read request** Format: U26 **Accumulative Memory Latency Count for the entire frame in 8xMedia clock cycles.** The accumulative memory latency count of all reference reads requested by motion compensative engine per frame. This number is used with **HCP Frame Motion Comp Read Count** to derive average memory latency.

HCP Memory Latency Count3

HCP_LAT_CT3 - HCP Memory Latency Count3

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E970h

This register stores the max and min memory latency counts reported on row-stored/bit-stream read requests. Max and current requests into memory sub-system engine.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** Format: U8 This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** Format: U8 This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the pre-fetch engine most likely hung waiting for read data to be returned from sub-system.
	15:8	**HCP row-stored/bit-stream read request - Max Latency Count in 8xMedia clock cycles** Format: U8 This field reports the maximum memory latency count on all row-stored/bit-stream reads requested by the memory pre-fetch engine.
	7:0	**HCP row-stored/bit-stream read request - Min Latency Count in 8xMedia clock cycles** Format: U8 This field reports the minimum memory latency count on all row-stored/bit-stream reads requested by the memory pre-fetch engine.

HCP Memory Latency Count4

HCP_LAT_CT4 - HCP Memory Latency Count4

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E974h

This register stores the accumulative memory latency count on row-stored/bit-stream read requests.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**HCP row-stored/bit-stream read request** Format: U32 **Accumulative Memory Latency Count for the entire frame in 8xMedia clock cycles.** The accumulative memory latency count of all row-stored/bit-stream reads requested by pre-fetch engine per frame. This number is used with **Frame row-stored/bit-stream memory read count** to derive average memory latency.

HCP Memory Latency Count5

HCP_LAT_CT5 - HCP Memory Latency Count5

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E978h

This register stores the max and min memory latency counts reported on PAK Object read requests. Max and current requests into memory sub-system engine.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** Format: U8 This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** Format: U8 This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the pre-fetch engine most likely hung waiting for read data to be returned from sub-system.
	15:8	**MFX PAK Object read request - Max Latency Count in 8xMedia clock cycles** Format: U8 This field reports the maximum memory latency count on all PAK Object reads requested by the memory pre-fetch engine.
	7:0	**MFX PAK Object read request - Min Latency Count in 8xMedia clock cycles** Format: U8 This field reports the minimum memory latency count on all PAK Object reads requested by the memory pre-fetch engine.

HCP Memory Latency Count6

HCP_LAT_CT6 - HCP Memory Latency Count6

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E97Ch

This register stores the max and min memory latency counts reported on Source Pixel read requests. Max and current requests into memory sub-system engine.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** Format: U8 This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** Format: U8 This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the pre-fetch engine most likely hung waiting for read data to be returned from sub-system.
	15:8	**FX Source Pixel read request - Max Latency Count in 8xMedia clock cycles** Format: U8 This field reports the maximum memory latency count on all Source Pixel reads requested by the memory pre-fetch engine. .
	7:0	**MFX Source Pixel read request - Min Latency Count in 8xMedia clock cycles** Format: U8 This field reports the minimum memory latency count on all Source Pixel reads requested by the memory pre-fetch engine.

HCP Picture Checksum cIdx0

HCP_PICTURE_CHECKSUM_CIDX0 - HCP Picture Checksum cIdx0

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E91Ch

- The HCP Picture Checksum cIdx0 register reports the 32-bit unsigned picture checksum for cIdx=0 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx0** Default Value: 0 Format: U32

HCP Picture Checksum cIdx1

HCP_PICTURE_CHECKSUM_CIDX1 - HCP Picture Checksum cIdx1	
Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E920h

- The HCP Picture Checksum cIdx1 register reports the 32-bit unsigned picture checksum for cIdx=1 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx1** Default Value: 0 Format: U32

HCP Picture Checksum cIdx2

HCP_PICTURE_CHECKSUM_CIDX2 - HCP Picture Checksum cIdx2

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E924h

- The HCP Picture Checksum cIdx2 register reports the 32-bit unsigned picture checksum for cIdx=2 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx2** Default Value: 0 Format: U32

HCP Qp Status Count

HCP_QP_STATUS_COUNT - HCP Qp Status Count		
Register Space:	MMIO: 0/2/0	
Source:	VideoCS	
Default Value:	0x00000000, 0x00000000	
Access:	RO	
Size (in bits):	64	
Trusted Type:	1	
Address:	1E9C0h	

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:0	**Cumulative QP** Format: U24 Cumulative QP for all LCU of a Frame (Can be used for computing average QP).
1	31:12	**Reserved**
	11:6	**Frame Max CU QP**
	5:0	**Frame Min CU QP**

HCP Reported Bitstream Output CABAC Bin Count Register

HCP_CABAC_BIN_COUNT_FRAME - HCP Reported Bitstream Output CABAC Bin Count Register

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E9ACh

This register stores the count of number of bins per frame.

DWord	Bit	Description
0	31:0	**HCP Cabac Bin Count** Default Value: 0 Format: U32 Total number of BINs in the bitstream output per frame from the encoder. This count is updated for every time the bin counter is incremented and its reset at image start.

HCP Slice Performance Count

HCP_SLICE_PERF_CNT - HCP Slice Performance Count

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E964h

This register stores the number of clock cycles spent decoding/encoding the current slice.

This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**Count** Format: U32 Total number of clocks between slice start and slice end. This count is incremented on crm_clk.

HCP Unit Done

HCP_UNIT_DONE - HCP Unit Done	
Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E9D8h

DWord	Bit	Description
0	31:25	**Reserved** Format: MBZ
	24:15	**Reserved** Format: MBZ
	14:11	**Reserved** Format: MBZ
	10:9	**Reserved**
	5	**HLC unit done** Format: U1
	4	**HLE unit done** Format: U1
	3	**HFQ unit done** Format: U1
	2	**HFT unit done** Format: U1
	1	**HRS unit done** Format: U1
	0	**HPO unit done** Format: U1

HDC TLB REQUEST CONTROL REGISTER

HDCTLB_REQ_CTRL - HDC TLB REQUEST CONTROL REGISTER		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
This is a basic register template		
DWord	**Bit**	**Description**
0	31:19	**Reserved** Default Value: 0000000000000b Access: RO

HDPORT_STATE

HDPORT_STATE

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	45050h-45053h
Name:	HDPORT State
ShortName:	HDPORT_STATE
Power:	PG0
Reset:	soft

This register is used to indicate when display resources have been pre-empted by hardware for the HDPORT feature. The usage is set during boot, before BIOS or software is active.
The list of DPLLs and DDIs in this register many not accurately reflect the total number of DPLLs and DDIs supported by display engine. HDPORT will not use any DPLL or DDI not listed here. It will not use any DPLL or DDI that is listed here, but not supported by the particular product or SKU.

Restriction
Restriction : Display software must not use resources that are marked as being used by HDPORT.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15	**DPLL2 Used** This field indicates whether DPLL 2 is being used by HDPORT. 0b – Not used 1b – Used
	14	**DPLL3 Used** This field indicates whether DPLL 3 is being used by HDPORT. 0b – Not used 1b – Used
	13	**DPLL1 Used** This field indicates whether DPLL 1 is being used by HDPORT. 0b – Not used 1b – Used

HDPORT_STATE

	12	**DPLL0 Used** This field indicates whether DPLL 0 is being used by HDPORT. **Value** / **Name** 0b – Not used 1b – Used
	11	**Spare 11**
	10	**Spare 10**
	9	**Spare 9**
	8	**DDI3 Type** This field indicates whether DDI 3 (DDI D) is being used in HDMI or DP mode by HDPORT. **Value** / **Name** 0b – DP 1b – HDMI
	7	**DDI3 Used** This field indicates whether DDI 3 (DDI D) is being used by HDPORT. **Value** / **Name** 0b – Not used 1b – Used
	6	**DDI2 Type** This field indicates whether DDI 2 (DDI C) is being used in HDMI or DP mode by HDPORT. **Value** / **Name** 0b – DP 1b – HDMI
	5	**DDI2 Used** This field indicates whether DDI 2 (DDI C) is being used by HDPORT. **Value** / **Name** 0b – Not used 1b – Used
	4	**DDI1 Type** This field indicates whether DDI 1 (DDI B) is being used by the HDPORT in HDMI or DP mode. **Value** / **Name** 0b – DP 1b – HDMI

HDPORT_STATE

	3	**DDI1 Used** This field indicates whether DDI 1 (DDI B) is being used by HDPORT. **Value** / **Name** 0b – Not used 1b – Used
	2	**DDI0 Type** This field indicates whether DDI 0 (DDI A and DDI E) is being used in HDMI or DP mode by HDPORT. **Value** / **Name** 0b – DP 1b – HDMI
	1	**DDI0 Used** This field indicates whether DDI 0 (DDI A and DDI E) is being used by HDPORT. **Value** / **Name** 0b – Not used 1b – Used
	0	**HDPORT Enabled** This field indicates whether HDPORT is enabled. **Value** / **Name** 0b – Disabled 1b – Enabled

Header Type

HDR2_0_2_0_PCI - Header Type

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	8
Address:	0000Eh

This register contains the Header Type of the IGD.

DWord	Bit	Description
0	7	**Multi Function Status** Default Value: 0b Access: RO Indicates if the device is a Multi-Function Device. The Value of this register is hardwired to 0, internal graphics is a single function.
	6:0	**Header Code** Default Value: 0000000b Access: RO This is a 7-bit value that indicates the Header Code for the IGD. This code is hardwired to the value 00h, indicating a type 0 configuration space format.

HEVC GAM Slave Counter High part

HEVC_GAM_SLAVE_CTR_H - HEVC GAM Slave Counter High part

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	04880h
Name:	HEVC slave counter high for VDBOX0
ShortName:	HEVC_VDBOX0_CTR_H
Address:	04884h
Name:	HEVC slave counter high for VDBOX1
ShortName:	HEVC_VDBOX1_CTR_H
Address:	04888h
Name:	HEVC slave counter high for VDBOX2
ShortName:	HEVC_VDBOX2_CTR_H
Address:	0488Ch
Name:	HEVC slave counter high for VDBOX3
ShortName:	HEVC_VDBOX3_CTR_H
Address:	04890h
Name:	HEVC slave counter high for VDBOX4
ShortName:	HEVC_VDBOX4_CTR_H
Address:	04894h
Name:	HEVC slave counter high for VDBOX5
ShortName:	HEVC_VDBOX5_CTR_H
Address:	04898h
Name:	HEVC slave counter high for VDBOX6
ShortName:	HEVC_VDBOX6_CTR_H
Address:	0489Ch
Name:	HEVC slave counter high for VDBOX7
ShortName:	HEVC_VDBOX7_CTR_H

DWord	Bit	Description
0	31:0	**HEVC GAM SLave Counter High** Default Value: 00000000h Access: R/W HEVC GAM Slave counter[63:32]

HEVC GAM Slave Counter Low part

HEVC_GAM_SLAVE_CTR_L - HEVC GAM Slave Counter Low part

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	04840h
Name:	HEVC slave counter low for VDBOX0
ShortName:	HEVC_VDBOX0_CTR_L
Address:	04844h
Name:	HEVC slave counter low for VDBOX1
ShortName:	HEVC_VDBOX1_CTR_L
Address:	04848h
Name:	HEVC slave counter low for VDBOX2
ShortName:	HEVC_VDBOX2_CTR_L
Address:	0484Ch
Name:	HEVC slave counter low for VDBOX3
ShortName:	HEVC_VDBOX3_CTR_L
Address:	04850h
Name:	HEVC slave counter low for VDBOX4
ShortName:	HEVC_VDBOX4_CTR_L
Address:	04854h
Name:	HEVC slave counter low for VDBOX5
ShortName:	HEVC_VDBOX5_CTR_L
Address:	04858h
Name:	HEVC slave counter low for VDBOX6
ShortName:	HEVC_VDBOX6_CTR_L
Address:	0485Ch
Name:	HEVC slave counter low for VDBOX7
ShortName:	HEVC_VDBOX7_CTR_L

DWord	Bit	Description
0	31:0	**HEVC GAM SLave Counter Low** Default Value: 00000000h Access: R/W HEVC GAM Slave counter[31:0]

HEVC Local APIC Retry Vector

HEVC_LAPIC_RETRY_VECT - HEVC Local APIC Retry Vector

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0D594h

Holds the 4 last retry interrupt vectors. The retry vector register holds the last 4 values acknowledged as an interrupt retry. Retries are errors in hardware and are not expected. HUCINT handles retries by logging the interrupt vector in this register. No interrupt is actually retried, and the interrupt stimulus will be lost if a retry occurs. The system will hang eventually. A 2-bit counter (starting at reset value of 0) is used to point to the slot/byte location from which to load the next retry vector (into the 4 available slots) in sequence. This means if a 5th retry vector shows up, it will be loaded into slot 0 again (as the counter wraps around), over-writing the retry vector which existed there in slot_0.

DWord	Bit	Description
0	31:24	**Vector Slot 3** Format: U8
	23:16	**Vector Slot 2** Format: U8
	15:8	**Vector Slot 1** Format: U8
	7:0	**Vector Slot 0** Format: U8

HEVC Microcontroller Header Info

HUC_UKERNEL_HDR_INFO - HEVC Microcontroller Header Info

Register Space:	MMIO: 0/2/0
Source:	VideoCS
Default Value:	0x00000000
Size (in bits):	32
Address:	0D014h
Access:	RO
Address:	FFE0D014h
Access:	R/W

The Address 0D014h is accessible in the MCI register space

DWord	Bit	Description
0	31:10	**Reserved** Format: MBZ
	9:2	**Kernel ID** Access: RO Format: U8 Kernel specified by the HUC_IMEM_STATE command.
	1	**Reserved** Format: MBZ
	0	**uKernel Header Valid** Access: RO Format: U1 A value of 1 indicates that the register contents are loaded successfully and Kernel ID field is valid.

HS Invocation Counter

HS_INVOCATION_COUNT - HS Invocation Counter	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02300h

This register stores the number of patch objects processed by the HS unit. E.g., A PATCHLIST_2 topology with 6 vertices would cause this counter to increment by 3 (there are 3 2-vertex patch objects in that topology).This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**HS Invocation Count UDW** Number of patch objects processed by the HS stage. Updated only when HS Enable and HS Statistics Enable are set in 3DSTATE_HS
	31:0	**HS Invocation Count LDW** Number of patch objects processed by the HS stage. Updated only when HS Enable and HS Statistics Enable are set in 3DSTATE_HS

IA32 MTRR FIX4K_C0000 High

MTRR_FIX4K_C0000_H - IA32 MTRR FIX4K_C0000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F13Ch	
Fixed MTRR to identify (C0000-C8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_C0000 Low

MTRR_FIX4K_C0000_L - IA32 MTRR FIX4K_C0000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F138h	
Fixed MTRR to identify (C0000-C8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_C8000 High

MTRR_FIX4K_C8000_H - IA32 MTRR FIX4K_C8000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F144h	
Fixed MTRR to identify (C8000-D0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_C8000 Low

MTRR_FIX4K_C8000_L - IA32 MTRR FIX4K_C8000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F140h	
Fixed MTRR to identify (C8000-D0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_D0000 High

MTRR_FIX4K_D0000_H - IA32 MTRR FIX4K_D0000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F14Ch	
Fixed MTRR to identify (D0000-D8000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_D0000 Low

MTRR_FIX4K_D0000_L - IA32 MTRR FIX4K_D0000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F148h	
Fixed MTRR to identify (D0000-D8000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_D8000 High

MTRR_FIX4K_D8000_H - IA32 MTRR FIX4K_D8000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F154h	
Fixed MTRR to identify (D8000-E0000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_D8000 Low

MTRR_FIX4K_D8000_L - IA32 MTRR FIX4K_D8000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F150h	
Fixed MTRR to identify (D8000-E0000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_E0000 High

MTRR_FIX4K_E0000_H - IA32 MTRR FIX4K_E0000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F15Ch	
Fixed MTRR to identify (E0000-E8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_E0000 Low

MTRR_FIX4K_E0000_L - IA32 MTRR FIX4K_E0000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F158h	
Fixed MTRR to identify (E0000-E8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_E8000 High

MTRR_FIX4K_E8000_H - IA32 MTRR FIX4K_E8000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F164h	
Fixed MTRR to identify (E8000-F0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_E8000 Low

MTRR_FIX4K_E8000_L - IA32 MTRR FIX4K_E8000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F160h	
Fixed MTRR to identify (E8000-F0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_F0000 High

MTRR_FIX4K_F0000_H - IA32 MTRR FIX4K_F0000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F16Ch	
Fixed MTRR to identify (F0000-F8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_F0000 Low

MTRR_FIX4K_F0000_L - IA32 MTRR FIX4K_F0000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F168h	
Fixed MTRR to identify (F0000-F8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_F8000 High

MTRR_FIX4K_F8000_H - IA32 MTRR FIX4K_F8000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F174h	
Fixed MTRR to identify (F8000-100000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_F8000 Low

MTRR_FIX4K_F8000_L - IA32 MTRR FIX4K_F8000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F170h	
Fixed MTRR to identify (F8000-100000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX16K_80000 High

MTRR_FIX16K_80000_H - IA32 MTRR FIX16K_80000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F12Ch	
Fixed MTRR to identify 512K-768K of the main memory (80000-A0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX16K_80000 Low

MTRR_FIX16K_80000_L - IA32 MTRR FIX16K_80000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F128h	
Fixed MTRR to identify 512K-768K of the main memory (80000-A0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#.0

IA32 MTRR FIX16K_A0000 High

MTRR_FIX16K_A0000_H - IA32 MTRR FIX16K_A0000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F134h	
Fixed MTRR to identify 768K-1024K of the main memory (A0000-C0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX16K_A0000 Low

MTRR_FIX16K_A0000_L - IA32 MTRR FIX16K_A0000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F130h	
Fixed MTRR to identify 768K-1024K of the main memory (A0000-C0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX64K_00000 High

MTRR_FIX64K_00000_H - IA32 MTRR FIX64K_00000 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F124h	
Fixed MTRR to identify 0-512K of the main memory (0-80000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX64K_00000 Low

MTRR_FIX64K_00000_L - IA32 MTRR FIX64K_00000 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F120h	
Fixed MTRR to identify 0-512K of the main memory (0-80000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR PHYSBASE0 High

MTRR_PHYSBASE0_H - IA32 MTRR PHYSBASE0 High

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F184h

Variable MTRR0

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE0 Low

MTRR_PHYSBASE0_L - IA32 MTRR PHYSBASE0 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F180h

Variable MTRR0

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE1 High

MTRR_PHYSBASE1_H - IA32 MTRR PHYSBASE1 High

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F194h

Variable MTRR1

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE1 Low

MTRR_PHYSBASE1_L - IA32 MTRR PHYSBASE1 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F190h

Variable MTRR1

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE2 High

MTRR_PHYSBASE2_H - IA32 MTRR PHYSBASE2 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1A4h	
Variable MTRR2		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE2 Low

MTRR_PHYSBASE2_L - IA32 MTRR PHYSBASE2 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1A0h

Variable MTRR2

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE3 High

MTRR_PHYSBASE3_H - IA32 MTRR PHYSBASE3 High

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1B4h

Variable MTRR3

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE3 Low

MTRR_PHYSBASE3_L - IA32 MTRR PHYSBASE3 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1B0h	
Variable MTRR3		
DWord	**Bit**	**Description**
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE4 High

MTRR_PHYSBASE4_H - IA32 MTRR PHYSBASE4 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1C4h	
Variable MTRR4		

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE4 Low

MTRR_PHYSBASE4_L - IA32 MTRR PHYSBASE4 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1C0h

Variable MTRR4

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh

IA32 MTRR PHYSBASE5 High

MTRR_PHYSBASE5_H - IA32 MTRR PHYSBASE5 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1D4h	
Variable MTRR5		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE5 Low

MTRR_PHYSBASE5_L - IA32 MTRR PHYSBASE5 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1D0h	
Variable MTRR5		
DWord	**Bit**	**Description**
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE6 High

MTRR_PHYSBASE6_H - IA32 MTRR PHYSBASE6 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1E4h	
Variable MTRR6		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 0000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE6 Low

MTRR_PHYSBASE6_L - IA32 MTRR PHYSBASE6 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1E0h

Variable MTRR6

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE7 High

MTRR_PHYSBASE7_H - IA32 MTRR PHYSBASE7 High

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F4h

Variable MTRR7

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE7 Low

MTRR_PHYSBASE7_L - IA32 MTRR PHYSBASE7 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F0h

Variable MTRR7

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE8 High

MTRR_PHYSBASE8_H - IA32 MTRR PHYSBASE8 High

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F204h

Variable MTRR8

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE8 Low

MTRR_PHYSBASE8_L - IA32 MTRR PHYSBASE8 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F200h

Variable MTRR8

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE9 High

MTRR_PHYSBASE9_H - IA32 MTRR PHYSBASE9 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F214h	
Variable MTRR9		

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE9 Low

MTRR_PHYSBASE9_L - IA32 MTRR PHYSBASE9 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F210h

Variable MTRR9

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSMASK0 High

MTRR_PHYSMASK0_H - IA32 MTRR PHYSMASK0 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F18Ch	
Variable MTRR0		

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK0 Low

MTRR_PHYSMASK0_L - IA32 MTRR PHYSMASK0 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F188h

Variable MTRR0

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK1 High

MTRR_PHYSMASK1_H - IA32 MTRR PHYSMASK1 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F19Ch	
Variable MTRR1		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK1 Low

MTRR_PHYSMASK1_L - IA32 MTRR PHYSMASK1 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F198h

Variable MTRR1

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK2 High

MTRR_PHYSMASK2_H - IA32 MTRR PHYSMASK2 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1ACh	
Variable MTRR2		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK2 Low

MTRR_PHYSMASK2_L - IA32 MTRR PHYSMASK2 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1A8h	
Variable MTRR2		
DWord	**Bit**	**Description**
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK3 High

MTRR_PHYSMASK3_H - IA32 MTRR PHYSMASK3 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1BCh	
Variable MTRR3		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK3 Low

MTRR_PHYSMASK3_L - IA32 MTRR PHYSMASK3 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1B8h	
Variable MTRR3		
DWord	**Bit**	**Description**
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK4 High

MTRR_PHYSMASK4_H - IA32 MTRR PHYSMASK4 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1CCh	
Variable MTRR4		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK4 Low

MTRR_PHYSMASK4_L - IA32 MTRR PHYSMASK4 Low		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1C8h	
Variable MTRR4		

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK5 High

MTRR_PHYSMASK5_H - IA32 MTRR PHYSMASK5 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1DCh	
Variable MTRR5		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK5 Low

MTRR_PHYSMASK5_L - IA32 MTRR PHYSMASK5 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1D8h

Variable MTRR5

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK6 High

MTRR_PHYSMASK6_H - IA32 MTRR PHYSMASK6 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1ECh	
Variable MTRR6		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK6 Low

MTRR_PHYSMASK6_L - IA32 MTRR PHYSMASK6 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1E8h

Variable MTRR6

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK7 High

MTRR_PHYSMASK7_H - IA32 MTRR PHYSMASK7 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F1FCh	
Variable MTRR7		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK7 Low

MTRR_PHYSMASK7_L - IA32 MTRR PHYSMASK7 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F8h

Variable MTRR7

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK8 High

MTRR_PHYSMASK8_H - IA32 MTRR PHYSMASK8 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F20Ch	
Variable MTRR8		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK8 Low

MTRR_PHYSMASK8_L - IA32 MTRR PHYSMASK8 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F208h

Variable MTRR8

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK9 High

MTRR_PHYSMASK9_H - IA32 MTRR PHYSMASK9 High		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F21Ch	
Variable MTRR9		
DWord	**Bit**	**Description**
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK9 Low

MTRR_PHYSMASK9_L - IA32 MTRR PHYSMASK9 Low

Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	0F218h

Variable MTRR9

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA Vertices Count

IA_VERTICES_COUNT - IA Vertices Count	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02310h

This register stores the count of vertices processed by VF. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**IA Vertices Count Report UDW** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)
	31:0	**IA Vertices Count Report LDW** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)

IDI Cacheable Register

IDICA - IDI Cacheable Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	09014h

Cacheable

DWord	Bit	Description
0	31:30	**LLCWBCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	29:28	**LLCPRFOCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	27:26	**LLCPCCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	25:24	**LLCPDCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	23:22	**CLFCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	21:20	**POCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	19:18	**ITMCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	17:16	**WCILFCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	15:14	**WILCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.

IDICA - IDI Cacheable Register

	13:12	**WCILCA** Access: R/W >NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	11:10	**WBMCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	9:8	**RFOCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	7:6	**PORINCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	5:4	**PRDCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b.
	3:2	**DRDCA** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.
	1:0	**CRDCA** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.

IDI Control register

IDICR - IDI Control register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000181	
Size (in bits):	32	
Address:	09008h	
DWord	**Bit**	**Description**
0	31:24	**Spares** Access: R/W ECO purposes and Reserved.
	23:22	**QOS setting for Frame Buffer Caching** Access: R/W The value determines the quality of service setting of Frame Buffers while being cached in Last Level Cache/eDRAM. GFX Driver has to set the value of QOS after enabling Frame Buffer caching and deciding which ways of LLC is allocated for which QOS values. 00 : QOS setting of 00 used for Frame Buffers 01 : QOS setting of 01 used for Frame Buffers 10 : QOS setting of 10 used for Frame Buffers 11 : QOS setting of 11 used for Frame Buffers
	21:16	**IDI HASH MASK** Access: R/W IDI HASH MASK: When a corresponding bit is set, the address line going into HASH for CBO ID calculation is forced to logic0. 21=> Address Bit[11] 20=> Address Bit[10] 19=> Address Bit[9] 18=> Address Bit[8] 17=> Address Bit[7] 16=> Address Bit[6] Note: It is required for GFX Driver to set [19:16] to 1 when eDRAM configuration is enabled. For Skylake, S/W is not needed to program this register as eDRAM is a memory side cache.
	15	**GFX Data regulation** Access: R/W BGF data regulation for incoming streams with chunkID detection.
	14:10	**Reserved**

IDICR - IDI Control register

	9	**RSVD** Access: RO
	8	**Push Write Enable** Default Value: 1b Access: R/W Push Write Enable: Push writes are a new mechanism to deliver write data to Uncore. It provides two advantages. 1) Reduced TAG pass requirements 2) Only way to allocate into eLLC without going thru LLC. The downside is the fact that push writes are weakly ordered which means a synchronizing event is required to guarantee consistency of data.
	7	**Snoop Request control** Default Value: 1b Access: R/W 1: Snoop is allowed only when there are no Pending response. 0: Means after every 24 u2c response we allow one snoop request to bypass.
	6:4	**LRUHint** Access: R/W 000b: No LRUHint command sent to uncore. It is reserved. 001b: If LRUHint is asserted from SQ with a read or write command, IDI dispatcher chooses to send an LlcPrefData. 101b: If LRUHint is asserted from SQ with a read or write command, IDI dispatcher chooses to send an LLCPrefCode command on the C2U request channel. 010b: If LRUHint is asserted from SQ with a read/write command, IDI dispatcher chooses to send an LlcPrefRFO command on the C2U request channel. 011b: If LRUHint is asserted from SQ with a read, IDI dispatcher chooses to send LlcPrefData command on the C2U request channel. If LRUHint is asserted from SQ with a write, IDI dispatcher chooses to send LlcPrefRFO command on the C2U request channel. 111b: If LRUHint is asserted from SQ with a read, IDI dispatcher chooses to send LLCPrefCode command on the C2U request channel. If LRUHint is asserted from SQ with a write, IDI dispatcher chooses to send LlcPrefRFO command on the C2U request channel.
	3	**RSVD** Access: RO
	2	**Resport 1 disable** Access: R/W 0: Default value - Both the Response ports on the BGF side are enabled. 1: Rsp Port1 Disable - Response Port1 is disable on the BGF Side.

IDICR - IDI Control register

	1:0	**SQ Grant Counter** Default Value: 01b Access: R/W SQ grant counter - 2-bit grant counter for SQ requests 00b: 1 grant. 01b: 2 grants. 10b: 4 grants. 11b: 8 grants.

IDI HASH Mask Register

DRBIDI3 - IDI HASH Mask Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	01948h

DWord	Bit	Description
0	31:10	**Reserved** Access: RO Reserved
	9:8	**Reserved**
	7:6	**Reserved** Access: RO Reserved
	5:0	**IDI HASH MASK** Access: R/W When corresponding MASK bit is set, the masked address bit going into HASH calculator is forced to be logic0. 21=> Address Bit[11] 20=> Address Bit[10] 19=> Address Bit[9] 18=> Address Bit[8] 17=> Address Bit[7] 16=> Address Bit[6] For Gen8 with 128MB eDRAM eLLC, bits[5:0] should be set to 001111 (matching 9008[21:16] IDI hash mask) In SKL, it is no longer needed for s/w to program this field given that eDRAM is a memory side cache

IDI Look up Register

IDILK2 - IDI Look up Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	08514h

IDI Look up Register

DWord	Bit	Description
0	31:30	**Spares** Access: R/W Lock
	28	**Colloc bit for Slice 5** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	27	**Direction bit for Slice 5** Access: R/W Lock In Half ring uncore topologies this indicates if the 1: Going Up. 0: Going Down.
	26	**Polarity bit for Slice 5** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even. 0 - Odd.
	25	**For Me for Slice 5** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	24	**Spares2** Access: R/W Lock Reserved for Slice 4.

IDILK2 - IDI Look up Register

	23	**Colloc bit for Slice 4** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	22	**Direction bit for Slice 4** Access: R/W Lock In Half ring uncore topologies this indicates if the 1: Going Up. 0: Going Down.
	21	**Polarity Bit for Slice 4** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even. 0 - Odd.
	20	**For Me bit for Slice 4** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	19	**Spare for Slice 3** Access: R/W Lock Reserved for Slice 3.
	18	**Colloc bit for Slice 3** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	17	**Direction bit for S3** Access: R/W Lock In Half ring uncore topologies this indicates if the 1: Going Up. 0: Going Down.
	16	**Polarity Bit for Slice 3** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination).

IDILK2 - IDI Look up Register

		1 - Even. 0 - Odd.
	15	**For Me Bit for Slice 3** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	14	**Spare for Slice 2** Access: R/W Lock Reserved for Slice 2.
	13	**Colloc bit for Slice 2** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	12	**Direction Bit for Slice 2** Access: R/W Lock In Half ring uncore topologies this indicates if the 1: Going Up. 0: Going Down.
	11	**Polarity Bit for Slice 2** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even. 0 - Odd.
	10	**For me Bit for Slice 2** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	9	**Spare for Slice 1** Access: R/W Lock Reserved for Slice 1.
	8	**Colloc Bit for Slice 1** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.

IDILK2 - IDI Look up Register

	7	**Direction Bit for Slice 1** Access: R/W Lock In Half ring uncore topologies this indicates if the 1: Going Up. 0: Going Down.
	6	**Polarity Bit for Slice 1** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even. 0 - Odd.
	5	**For Me Bit for Slice 1** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	4	**Spare for Slice 0** Access: R/W Lock Reserved for Slice 0.
	3	**Colloc Bit for Slice 0** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	2	**Direction Bit in Slice0** Access: R/W Lock Direction bit for Slice0: In Half ring uncore topologies this indicates if the request needs to be driven on the Up going (1) or the down (0) going ring direction. For Full ring it indicates Clock-wise (1) or counter clock-wise directions. 1: Going Up. 0: Going Down.
	1	**Polarity Bit for Slice 0** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even.

IDILK2 - IDI Look up Register		
		0 - Odd.
	0	**For Me bit for Slice0** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).

IDILook up Table register

IDILK1 - IDILook up Table register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	08510h

IDI Look Up register I

DWord	Bit	Description
0	31:21	**Spares** Access: R/W Lock
	20:16	**GT Logical ID** Access: R/W Lock Logical ID for GT.
	15:14	**Spares1** Access: R/W Lock Reserved for SA slice.
	13	**Colloc bit for SA Slice** Access: R/W Lock Co-located indicates that the Collocated Cbo should receive this request.
	12	**Direction Bit for SA** Access: R/W Lock In Half ring uncore topologies this indicates if the request needs to be driven on the Up going (1) or the down (0) going ring direction. For Full ring it indicates Clock-wise (1) or counter clock-wise directions. 1: Going Up. 0: Going Down.
	11	**Polarity bit for SA Slice** Access: R/W Lock Polarity based on the current core Slice ID and the Destination Cbo ID - should this request be sent to the rings in Even or Odd cycles (this is basically the Distance between the source and destination). 1 - Even. 0 - Odd.

IDILK1 - IDILook up Table register

	10	**For Me bit for SA** Access: R/W Lock The next slice the Target of this request (MyNeigbourId == DestCboId).
	9:5	**Number of LLC SA Slices** Access: R/W Lock Number of Slice information in the system. This register contains the number of LLC cache slices on the RING. Default: 0000b.
	4:0	**Colocated Slice ID for GT** Access: R/W Lock This register contains the ID of the slice that is servicing GT's co-located cycles. The default is for slice0 to service GT.

IDI MESSAGES

IDIMSG - IDI MESSAGES

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	08500h

IDI Message Register

DWord	Bit	Description
0	31:16	**Mask Bits** Access: RO Reserved.
	15:13	**RSVD** Access: RO
	12	**MCHECK COMPLETE** Access: R/W iMPH writes to this bit to initiate MCHECK COMPLETE Routine (PPPE flow). MBCunit will clear this bit once the PPPE flow is complete
	11	**Spare** Access: R/W Spare Messaging Bit with self-clear.
	10	**MBC Busy ACK** Access: R/W 1 - Busy ACK from GPMunit(Non-Idle). 0 - Non Busy ACK from Gpmunit (Idle). This bit is valid only if 26th Bit is set.
	9	**Reserved**
	8	**Reserved**
	7	**RSVD** Access: RO
	6	**Request to Block IDI** Access: R/W Block and Unblock IDI Request - usually done during CPD Entry and Exits. This is valid only if 22nd bit is set. Block IDI - CPD Entry = 1. Unblock IDI CPD Exit = 0.

IDIMSG - IDI MESSAGES

	5	**Unblock MMIO ack** Access: R/W Unblock MMIO ACK coming from SA. This is valid only if 21st bit is set.
	4	**Mbcunit Arbitration request/Release ACK** Access: R/W Arbitration request is sent during the MAE update. The ack is received from GPMunit. This is valid only if 20th bit is set. Arb req ack = 1. Arb release ack = 0.
	3	**IDI Shutdown request** Access: R/W IDI Shutdown Request from GPM to MBCunit. This is valid only if the 19th bit is set.
	2	**IDI Wakeup Message** Access: R/W IDI wakeup message from PM to MBCunit. This is valid only if 18th bit is set.
	1	**Credit Active De-assertreq ACK** Access: R/W Credit Active De-assertreq ACK - GPMunit sends to the MBCunit. This is valid only if the 17th bit of this register is set.
	0	**Reserved**

IDI Self Snoop Register

IDISLFSNP - IDI Self Snoop Register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	09018h	
Cacheable		
DWord	**Bit**	**Description**
0	31:30	**LLCWBSNP** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.
	29:28	**LLCPRFOSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	27:26	**LLCPCSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	25:24	**LLCPDSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	23:22	**CLFCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	21:20	**POCA** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	19:18	**ITMSNP** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default.

IDISLFSNP - IDI Self Snoop Register

		01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.
	17:16	**WCILFSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	15:14	**WILSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	13:12	**WCILSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	11:10	**WBMSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	9:8	**RFOSNP** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.
	7:6	**PORINSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	5:4	**PRDSNP** Access: R/W NOTE - THIS SHOULD ALWAYS BE PROGRAMMED TO 00b. CANNOT BE FLEXED.
	3:2	**DRDSNP** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.

IDISLFSNP - IDI Self Snoop Register

	1:0	**CRDSP** Access: R/W 00b: Whatever the logic decides (so force feature is disabled) - Default. 01b: Always drive 0. 10b: Always drive 1. 11b: Reserved.

Idle Switch Delay

IDLEDLY - Idle Switch Delay

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0223Ch-0223Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_RCSUNIT
Address:	1223Ch-1223Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VCSUNIT0
Address:	1A23Ch-1A23Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VECSUNIT
Address:	1C23Ch-1C23Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VCSUNIT1
Address:	2223Ch-2223Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_BCSUNIT

The IDLEDLY register contains an Idle Delay field which specifies eight times the time stamp base units allowed for command streamer to wait before a context is switched out leading to IDLE state in Execlist mode, i.e following this context switch there is no active element available in HW to execute. Refer "Time Stamp Bases[SKL+]" subsection in Power Management chapter for time stamp base unit granularity. Example: An IDLE Delay count of "2" with Time stamp base unit value of 80ns would mean an idle delay wait of 1280ns (2*8*80). A default value of 0, means that by default, there is no restriction to wait on a context switch leading to IDLE. This register has no significance when Execlists are not enabled.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:0	**IDLE Delay** Format: U21 Eight times the time stamp base units allowed. Refer "Time Stamp Bases[SKL]" subsection in Power Management chapter for time stamp base unit granularity. Example: An IDLE Delay count of "2" with Time stamp base unit value of 80ns would mean an idle delay wait of 1280ns (2*8*80).

Indirect Context Offset Pointer

INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000980 [KBL]
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	021C8h-021CBh
Name:	Indirect Context Offset Pointer
ShortName:	INDIRECT_CTX_OFFSET_RCSUNIT
Address:	121C8h-121CBh
Name:	Indirect Context Offset Pointer
ShortName:	INDIRECT_CTX_OFFSET_VCSUNIT0
Address:	1A1C8h-1A1CBh
Name:	Indirect Context Offset Pointer
ShortName:	INDIRECT_CTX_OFFSET_VECSUNIT
Address:	1C1C8h-1C1CBh
Name:	Indirect Context Offset Pointer
ShortName:	INDIRECT_CTX_OFFSET_VCSUNIT1
Address:	221C8h-221CBh
Name:	Indirect Context Offset Pointer
ShortName:	INDIRECT_CTX_OFFSET_BCSUNIT

This register is used to program the offset where commands RCS_INDIRECT_CTX points to will be executed as part of engine context restore.

Programming Notes	Source
BlitterCS/VideoCS/VideoCS2/VideoEnhancementCS: This register functionality is not supported and must not be programmed for these command streamers.	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Offset of Indirect CS context must be always programmed to a command boundary and cacheline boundary inside the context image.	
Indirect context pointer itself is restored during context restore and hence Indirect Context Offset must not be programmed with value less than 0x5.	

DWord	Bit	Description	
0	31:16	**Reserved**	
		Format:	MBZ

INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

	15:6	**Offset of Indirect CS Context** Format: U10 This is the cache line offset for the Indirect CS context. This defaults to execute between CS and SVG context. It is not valid to program this to a value that is greater or equal to the starting offset for RS context. If context must be programmed at the end of engine context then program then use BB_PER_CTX_PTR. **Value** / **Name** 26h / **[Default]**
	5:0	**Reserved** Format: MBZ

Indirect Context Pointer

INDIRECT_CTX - Indirect Context Pointer

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Address:	021C4h-021C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_RCSUNIT

Address:	121C4h-121C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VCSUNIT0

Address:	1A1C4h-1A1C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VECSUNIT

Address:	1C1C4h-1C1C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VCSUNIT1

Address:	221C4h-221C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_BCSUNIT

This register is used to program the indirect address to be executed between CS and SVG engine context if enabled. This will only get executed due to regular context save/restore and not during power restore. This register is part of the execution list context and will be executed per context. Only supported if execution list is enabled. There is no preempting workloads within this context.

Programming Notes	Source
BlitterCS/VideoCS/VideoCS2/VideoEnhancementCS: This register functionality is not supported and must not be programmed for these command streamers.	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
The following commands are not supported within Render CS indirect context: **Command Name** MI_WAIT_FOR_EVENT MI_SEMAPHORE_SIGNAL MI_ARB_CHECK MI_RS_CONTROL	RenderCS

INDIRECT_CTX - Indirect Context Pointer

MI_REPORT_HEAD		
MI_URB_ATOMIC_ALLOC		
MI_SUSPEND_FLUSH		
MI_TOPOLOGY_FILTER		
MI_RS_CONTEXT		
MI_SET_CONTEXT		
MI_URB_CLEAR		
MI_SEMAPHORE_WAIT in register poll mode is supported.		
MI_BATCH_BUFFER_START		
MI_CONDITIONAL_BATCH_BUFFER_END		
MEDIA_OBJECT_WALKER		
GPGPU_WALKER		
3DPRIMITIVE		
3DSTATE_BINDING_TABLE_POINTERS_VS		
3DSTATE_BINDING_TABLE_POINTERS_HS		
3DSTATE_BINDING_TABLE_POINTERS_DS		
3DSTATE_BINDING_TABLE_POINTERS_GS		
3DSTATE_BINDING_TABLE_POINTERS_PS		
3DSTATE_GATHER_CONSTANT_VS		
3DSTATE_GATHER_CONSTANT_GS		
3DSTATE_GATHER_CONSTANT_HS		
3DSTATE_GATHER_CONSTANT_DS		
3DSTATE_GATHER_CONSTANT_PS		
3DSTATE_DX9_CONSTANTF_VS		
3DSTATE_DX9_CONSTANTF_HS		
3DSTATE_DX9_CONSTANTF_DS		
3DSTATE_DX9_CONSTANTF_GS		
3DSTATE_DX9_CONSTANTF_PS		
3DSTATE_DX9_CONSTANTI_VS		
3DSTATE_DX9_CONSTANTI_HS		
3DSTATE_DX9_CONSTANTI_DS		
3DSTATE_DX9_CONSTANTI_GS		
3DSTATE_DX9_CONSTANTI_PS		
3DSTATE_DX9_CONSTANTB_VS		

INDIRECT_CTX - Indirect Context Pointer

3DSTATE_DX9_CONSTANTB_HS
3DSTATE_DX9_CONSTANTB_DS
3DSTATE_DX9_CONSTANTB_GS
3DSTATE_DX9_CONSTANTB_PS
3DSTATE_DX9_LOCAL_VALID_VS
3DSTATE_DX9_LOCAL_VALID_DS
3DSTATE_DX9_LOCAL_VALID_HS
3DSTATE_DX9_LOCAL_VALID_GS
3DSTATE_DX9_LOCAL_VALID_PS
3DSTATE_DX9_GENERATE_ACTIVE_VS
3DSTATE_DX9_GENERATE_ACTIVE_HS
3DSTATE_DX9_GENERATE_ACTIVE_DS
3DSTATE_DX9_GENERATE_ACTIVE_GS
3DSTATE_DX9_GENERATE_ACTIVE_PS
3DSTATE_BINDING_TABLE_EDIT_VS
3DSTATE_BINDING_TABLE_EDIT_GS
3DSTATE_BINDING_TABLE_EDIT_HS
3DSTATE_BINDING_TABLE_EDIT_DS
3DSTATE_BINDING_TABLE_EDIT_PS
3DSTATE_CONSTANT_VS
3DSTATE_CONSTANT_GS
3DSTATE_CONSTANT_PS
3DSTATE_CONSTANT_HS
3DSTATE_CONSTANT_DS
MI_BATCH_BUFFER_END

DWord	Bit	Description
0	31:6	**Indirect CS Context Address** Format: GraphicsAddress[31:6] Pointer to the Context in memory to be executed as a batch.
	5:0	**Size of Indirect CS Context** Format: U6 This is the size of the Indirect Context for CS. This size supports up to 63 cache lines worth of commands where a cache line is 64B. If programmed to zero then the indirect fetch of the CS context is disabled. Value: [0,63] Name:

INF unit Level Clock Gating Control 9560

INFCGCTL9560 - INF unit Level Clock Gating Control 9560

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x000000FF
Size (in bits):	32
Address:	09560h

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	31:8	**Reserved** Access: R/W Reserved
	7	**CPMAunit Clock Gating Disable** Default Value: 1b Access: R/W CPMAunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**GTFSunit Clock Gating Disable** Default Value: 1b Access: R/W GTFSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**RPMunit Clock Gating Disable** Default Value: 1b Access: R/W RPMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

INFCGCTL9560 - INF unit Level Clock Gating Control 9560

4	**MBGFUCunit Clock Gating Disable** Default Value: 1b Access: R/W MBGFUCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
3	**CGPSFunit Clock Gating Disable** Default Value: 1b Access: R/W CGPSFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
2	**MDRBunit Clock Gating Disable** Default Value: 1b Access: R/W MDRBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
1	**MGSRunit Clock Gating Disable** Default Value: 1b Access: R/W MGSRunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
0	**MRCunit Clock Gating Disable** Default Value: 1b Access: R/W MRCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Instruction Parser Mode Register

INSTPM - Instruction Parser Mode Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1

Address:	020C0h-020C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_RCSUNIT

Address:	120C0h-120C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VCSUNIT0

Address:	1A0C0h-1A0C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VECSUNIT

Address:	1C0C0h-1C0C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VCSUNIT1

Address:	220C0h-220C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_BCSUNIT

The INSTPM register is used to control the operation of the Instruction Parser. Certain classes of instructions can be disabled (ignored) - often useful for detecting performance bottlenecks. Also, Synchronizing Flush operations can be initiated - useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes

- If an instruction type is disabled, the parser will read those instructions but not process them.
- Error checking will be performed even if the instruction is ignored.
- All Reserved bits are implemented.
- This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask

INSTPM - Instruction Parser Mode Register

		Masks: These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15:14	**Reserved** Format: MBZ
	13	**Reserved** Format: MBZ
	12	**Reserved** Format: MBZ
	11	**CLFLUSH Toggle** Source: RenderCS Access: RO Format: U1 This bit changes polarity each time the MI_CLFLUSH command completes. This bit is Read Only.
	11	**Reserved** Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ
	10	**Implied Atomic Fences To Write Fences** Format: U1 If set, all implied atomic fences generated by Render Command Streamer during various operations (Flushes, Context Switch) are converted to regular write fences. Setting this bit will affect data ordering functionalityf. When reset HW behaves as expected.
	9:0	**Reserved** Format: MBZ

Internal GAM State

INTSTATE - Internal GAM State		
Register Space:	MMIO: 0/2/0	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	040C0h	
DWord	**Bit**	**Description**
0	31:0	**Reserved**

Interrupt Line

INTRLINE_0_2_0_PCI - Interrupt Line	
Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	8
Address:	0003Ch

This register is used to communicate interrupt line routing information. The device itself does not use this value, rather it is used by device drivers and operating systems to determine priority and vector information.

DWord	Bit	Description
0	7:0	**Interrupt Connection** Default Value: 00000000b Access: R/W Used to communicate interrupt line routing information. POST software writes the routing information into this register as it initializes and configures the system. The value in this register indicates to which input of the system interrupt controller the device's interrupt pin is connected.

Interrupt Mask Register

IMR - Interrupt Mask Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32

Address:	020A8h-020ABh
Name:	Interrupt Mask Register
ShortName:	IMR_RCSUNIT

Address:	120A8h-120ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VCSUNIT0

Address:	1A0A8h-1A0ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VECSUNIT

Address:	1C0A8h-1C0ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VCSUNIT1

Address:	220A8h-220ABh
Name:	Interrupt Mask Register
ShortName:	IMR_BCSUNIT

The IMR register is used by software to control which Interrupt Status Register bits are masked or unmasked. Unmasked bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. Masked bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** Format: InterruptMask[32] Refer to the Interrupt Control Register section for bit definitions. This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR. Reserved bits in the Interrupt Control Register are RO.

Value	Name	Description
FFFF FFFFh	**[Default]**	
0h	Not Masked	Will be reported in the IIR
1h	Masked	Will not be reported in the IIR

Interrupt Pin

INTRPIN_0_2_0_PCI - Interrupt Pin		
Register Space:	PCI: 0/2/0	
Source:	BSpec	
Default Value:	0x00000001	
Size (in bits):	8	
Address:	0003Dh	
This register tells which interrupt pin the device uses.		
DWord	**Bit**	**Description**
0	7:0	**Interrupt Pin** Default Value: 00000001b Access: RO As a single function device, the IGD specifies INTA# as its interrupt pin. Hardwired to 01h = INTA#.

I/O Base Address

IOBAR_0_2_0_PCI - I/O Base Address

Register Space:	PCI: 0/2/0
Source:	BSpec
Default Value:	0x00000001
Size (in bits):	32
Address:	00020h

This register provides the Base offset of the I/O registers within Device #2. Bits 15:6 are programmable allowing the I/O Base to be located anywhere in 16bit I/O Address Space. Bits 2:1 are fixed and return zero; bit 0 is hardwired to a one indicating that 8 bytes of I/O space are decoded. Access to the 8Bs of IO space is allowed in PM state D0 when IO Enable (PCICMD bit 0) set. Access is disallowed in PM states D1-D3 or if IO Enable is clear or if Device #2 is turned off or if Internal graphics is disabled through the fuse or fuse override mechanisms.
Note that access to this IO BAR is independent of VGA functionality within Device #2.
If accesses to this IO bar is allowed then all 8, 16 or 32 bit IO cycles from IA cores that falls within the 8B are claimed.

DWord	Bit	Description
0	15:6	**IO Base Address** Default Value: 0000000000b Access: R/W Set by the OS, these bits correspond to address signals [15:6].
	5:3	**Reserved** Format: MBZ
	2:1	**Memory Type** Default Value: 00b Access: RO Hardwired to 0s to indicate 32-bit address.
	0	**Memory/IO Space** Default Value: 1b Access: RO Hardwired to "1" to indicate IO space.

IPC PER SUBSLICE

EUMETRICS_EVENT1 - IPC PER SUBSLICE	
Register Space:	MMIO: 0/2/0
Default Value:	0x00000000
Size (in bits):	32
Address:	00D90h

This register mirrors an accumulating count for EU Metric Event1.
It is enabled by configuration bits in GPMunit and SPMunits.
Note that count is never cleared and delta should be calculated by sampling the initial register value at the start and subtracting that value off from the final sample value.

DWord	Bit	Description
0	31:0	**EU Metric Event Count** Access: RO

KCR GAM slave counter High part

KCR_CTR_SLAVE_H - KCR GAM slave counter High part		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0482Ch	
DWord	**Bit**	**Description**
0	31:0	**KCR Slave Counter High** Default Value: 00000000h Access: R/W Slave High counter[63:32] for KCR

KCR GAM slave counter Low part

KCR_CTR_SLAVE_L - KCR GAM slave counter Low part		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04828h	
DWord	**Bit**	**Description**
0	31:0	**KCR Slave Counter Low** Default Value: 00000000h Access: R/W Slave Low counter[31:0] for KCR

L3 Bank Status

L3STAT - L3 Bank Status

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B128h

L3 Status register

DWord	Bit	Description
0	31	**L3 Fill Access Status bit** Access: RO This register is Hardware Set and Clear. Set condition: set when the first command is seen on LTCC-LTCD interface. Reset condition: reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface. Reset condition: This Flag will be reset only if we have atleast 1 modified line in the cache written by DC client.
	30	**Texture access Status bit** Access: RO This register is Hardware Set and Clear Set condition : set when the first command is seen on LTCC-LTCD interface. Reset condition : reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface.
	29	**Constant access Status bit** Access: RO This register is Hardware Set and Clear Set condition : set when the first command is seen on LTCC-LTCD interface. Reset condition : reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface.
	28	**State access Status bit** Access: RO This register is Hardware Set and Clear Set condition : set when the first command is seen on LTCC-LTCD interface. Reset condition : reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface.
	27	**EU data traffic access Status bit** Access: RO This register is Hardware Set and Clear Set condition : set when the first command is seen on LTCC-LTCD interface. Reset condition : reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface.

L3STAT - L3 Bank Status		
	26	**IA coherent access Status bit** Access: RO This register is Hardware Set and Clear Set condition : set when the first command is seen on LTCC-LTCD interface. Reset condition : reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface.
	25:0	**Reserved** Access: RO

L3 Control Register

L3CNTLREG - L3 Control Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000 [KBL]
Access:	R/W
Size (in bits):	32
Address:	07034h

Programming Notes

The L3 allocation programming should assign all ways of the cache with no left over ways. Refer to L3 section for the recommended settings.

Any L3 configuration change that reduces the data cache allocation when strong IA coherency is used requires the full flush of L3 prior to the programming update.
An explicit or implicit flush of L3 (DC Flush) through the command streamer doesn't result in flushing/invalidating the IA Coherent lines from L3. However this can be achieved by setting the "**Pipe line flush Coherent lines**" control bit in the "L3SQCREG4" register.

DWord	Bit	Description
0	31:25	**All L3 Client Pool** Access: R/W Number of ways allocated for the all client pool. This is a combined pool for all clients. Value: 30h — Name: **[Default]** **Programming Notes** When this field is non-zero, **DC Way Assignment** and **Read Only Client Pool** should be 0KB.
	24:18	**DC Way Assignment** Access: R/W Number of ways allocated for DC. Note this allocation is only for DC data types. **Programming Notes** Note: This field must be 0KB if All L3 Client Pool is non-zero. **Please refer to L3 HAS for valid programming values**
	17:11	**Read Only Client Pool** Access: R/W Number of ways allocated for Read Only L3 clients. This is a combined pool for all Read Only clients. **Programming Notes** Note: This field must be 0KB if All L3 Client Pool is non-zero. **Please refer to L3 HAS for valid programming values**

L3CNTLREG - L3 Control Register

10 Reserved

Access:	R/W
Format:	PBC

9 Error Detection Behavior Control

Access:	R/W
Format:	Enable

The L3 error detection can be enabled to hang the GPU on a non-recoverable error due to SER type events. Such option will be used when corresponding context has data consistency requirements.

Once error detection is enabled, s/w has to initialize URB or SLM to all 0's (based on usage model) prior to execution of the workload. Initialization is required to clean up the error detection logic and syndrome tracking.

Value	Name	Description
0h	**[Default]**	RTL does not hang on parity errors or double bit error
1h		RTL enforces a hang on parity errors or double bit error

8 GPGPU L3 Credit Mode Enable

Access:	R/W
Format:	Enable

This bit is required to be enabled under GPGPU workloads to provide the MAX latency coverage from L3 cache. It will override the registers 0xB100[18:14] and 0xB100[23:19], to 0 and the maximum value respectively.

7:1 URB Allocation

Access:	R/W

Number of ways allocated for URB usage

Value	Name
30h	**[Default]**

Programming Notes
Please refer to L3 HAS for valid programming values. At least one way needs to be programmed in L3 space.

0 SLM Mode Enable

Access:	R/W
Format:	Enable

When enabled, a 64KB (per bank) region of L3 is reserved for SLM.

L3 Control Register1

L3CNTLREG1 - L3 Control Register1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x8007FFF0 [KBL]
Size (in bits):	32
Address:	0B10Ch

DWord	Bit	Description
0	31:28	**Data Fifo Depth Control** Access: R/W Data Fifo Depth Control (TS mode). Value cannot be zero for normal operation. lbcf_csr_lc_datafifo_depth[3:0]. Value: 1000b — Name: **[Default]**
	27:24	**Data Clock off time** Access: R/W Description: Data Clock off time (DATACLKOFF): Data Clock off time - Data block is shut off after these many number of clocks programmed in this register bits. lbcf_csr_lc_dataclkoff_time[3:0]. Min value to be 4'h0100. It should be between 4'h4 : 4'hf. Value: 1100b — Name: **[Default]**
	23:20	**TAG CLK OFF TIME** Access: R/W Description: TAG CLK OFF TIME (TAGCLKOFF): TAG Clock Off time. This is the time, which Clock gating Logic checks before it turns off the clock. lbcf_csr_lc_tagclkoff_time[3:0]. Value can be between 4'h4 - 4'hf.

L3CNTLREG1 - L3 Control Register1

Value	Name
0100b	**[Default]**

19 **L3 Aging Disable Bit**

Default Value:	0b
Access:	R/W

L3 Aging Disable Bit (L3AGDIS):
Aging Disable.
lbcf_csr_lc_agingdis.

18:15 **Fill aging**

Default Value:	1111b
Access:	R/W

Fill aging (L3AGF):
Aging Counter for Fill.
lbcf_csr_lc_fill_aging_cnt[3:0].
If bit B103.19 is 0 then this register value has to be nonzero.

14:11 **Aging Counter for Read 1 Port**

Default Value:	1111b
Access:	R/W

Aging Counter for Read 1 Port (L3AGR1):
Aging Counter for Read 1 Port.
lbcf_csr_lc_r1_aging_cnt[3:0].
If bit B103.19 is 0 then this register value has to be nonzero.

10:7 **L3 Aging Counter for R0**

Default Value:	1111b
Access:	R/W

L3 Aging Counter for R0 (L3AGR0):
Aging Counter for R0 Port.
lbcf_csr_lc_r0_aging_cnt[3:0].
If bit B103.19 is 0 then this register value has to be nonzero.

6:4 **L3 Aging Counter for SNOOP**

Default Value:	111b
Access:	R/W

L3 Aging Counter for SNOOP:
Aging Counter for Snoop Port.
lbcf_csr_lc_snp_aging_cnt[3:0].

L3CNTLREG1 - L3 Control Register1

	3:0	**Reserved** Default Value: 0000b Access: RO Reserved.

L3 LRA 0

L3_LRA_0 - L3 LRA 0		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x0E037C00	
Size (in bits):	32	
Exists If:	Device[Platform] == 'Client'	
Address:	04A10h	
DWord	**Bit**	**Description**
0	31:30	**L3** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min** Default Value: 0011100000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max** Default Value: 0011011111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 0 GPGPU

L3_LRA_0_GPGPU - L3 LRA 0 GPGPU

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x05013C00
Size (in bits):	32
Address:	04DD0h

DWord	Bit	Description
0	31:30	**L3 GPGPU** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min GPGPU** Default Value: 0001010000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max GPGPU** Default Value: 0001001111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min GPGPU** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 1

L3_LRA_1 - L3 LRA 1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x67F701BF	
Size (in bits):	32	
Exists If:	Device[Platform] == 'Client'	
Address:	04A14h	
DWord	**Bit**	**Description**
0	31:30	**DC** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max** Default Value: 1001111111b Access: R/W Maximum value of programmable LRA2.
	19:10	**L3 LRA2 Min** Default Value: 0111000000b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max** Default Value: 0110111111b Access: R/W Maximum value of programmable LRA1.

L3 LRA 1 GPGPU

L3_LRA_1_GPGPU - L3 LRA 1 GPGPU

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x67F8721B
Size (in bits):	32
Address:	04DD4h

DWord	Bit	Description
0	31:30	**DC GPGPU** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max GPGPU** Default Value: 1001111111b Access: R/W Maximum value of programmable LRA2.
	19:10	**L3 LRA2 Min GPGPU** Default Value: 1000011100b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max GPGPU** Default Value: 1000011011b Access: R/W Maximum value of programmable LRA1.

L3 LRA 2

L3_LRA_2 - L3 LRA 2		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000002	
Size (in bits):	32	
Exists If:	Device[Platform] == 'Client'	
Address:	04A18h	

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 LRA 2 GPGPU

L3_LRA_2_GPGPU - L3 LRA 2 GPGPU

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000002
Size (in bits):	32
Address:	04DD8h

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture GPGPU** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 LRA 0 3D

L3_LRA_0_3D - L3 LRA 0 3D		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x05013C00	
Size (in bits):	32	
Address:	04A10h	
DWord	**Bit**	**Description**
0	31:30	**L3 3D** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min 3D** Default Value: 0001010000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max 3D** Default Value: 0001001111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min 3D** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 1 3D

L3_LRA_1_3D - L3 LRA 1 3D

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x66F2D0B3 [KBL]
Size (in bits):	32
Address:	04A14h

DWord	Bit	Description
0	31:30	**DC 3D** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max 3D** Default Value: 1001101111b Access: R/W Maximum value of programmable LRA2.
	19:10	**L3 LRA2 Min 3D** Default Value: 0010110100b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max 3D** Default Value: 0010110011b Access: R/W Maximum value of programmable LRA1.

L3 LRA 2 3D

L3_LRA_2_3D - L3 LRA 2 3D		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000002	
Size (in bits):	32	
Address:	04A18h	
DWord	**Bit**	**Description**
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture 3D** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 LRA 2 3D

L3_LRA_2_3D - L3 LRA 2 3D

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x009FE70E
Size (in bits):	32
Address:	04A18h

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00000000b Access: RO
	23:14	**L3 LRA3 Max 3D** Default Value: 1001111111b Access: R/W Maximum value of programmable LRA2. If L3LRA3Min_3D == L3LRA3Max_3D , GATR LRA is disabled, GATR cycles are mapped to L3LRA0 If L3LRA3Min_3D == L3LRA3Max_3D , GATR LRA is disabled, L3LRA2Max_3D will default to L3LRA3Max_3D to reuse GATR entries
	13:4	**L3 LRA3 Min 3D** Default Value: 1001110000b Access: R/W Minimum value of programmable LRA3.
	3:2	**GATR_3D** Default Value: 11b Access: R/W Which LRA should GATR use.
	1:0	**Texture 3D** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 SLM Register

L3SLMREG - L3 SLM Register

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x40000000
Size (in bits):	32
Address:	0B110h

DWord	Bit	Description
0	31	**Disable Periodic SLM/SQ slot allocation** Default Value: 0b Access: R/W Disable Periodic SLM/SQ slot allocation: When cfg_lslm_livelock_fairarb_dis=1 lslm unit always has the higher priority and lslm_lsqc_block to lsqcunit is asserted as long as there are requests in SLM FIFO. lbcf_csr_lslm_livelock_fairarb_dis.
	30:26	**LSLM_SQ_PENDING_MAX** Default Value: 10000b Access: R/W If lslmunit has read data to be sent to lcbrunit this cfg register specifies the maximum number of clocks for which LSLMunit can block SQ request from being sent o lcbrunit. Default value = 8. Value cannot be zero. lbcf_csr_lslm_sqpend_max[4:0].
	25	**LSLM address disable** Default Value: 0b Access: R/W 0 - Enable b2b addr matching fix. lslmunit should not block the cycle in fifo if there is a match in the pipeline. 1 - Disable b2b addr matching fix. lslmunit should block the cycle in fifo if there is a match in the pipeline. lbcf_csr_lslm_same_addr_dis. Default = 0.
	24:0	**Reserved** Access: RO

L3 SQC register 4

L3SQCREG4 - L3 SQC register 4		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x40400000 [KBL]	
Size (in bits):	32	
Address:	0B118h	

DWord	Bit	Description
0	31	**Reserved**
	30	**L3SQ URB Read CAM Match Disable** Default Value: 1b Access: R/W L3SQ URB Read CAM Match Disable (SQURBRDCAMDIS): Disables the L3SQ Cam Match ability for URB Reads. By disabling, this allows a performance mode where URB reads are not dependent upon one another but only on any previous URB writes to the same address. This allows many URB reads to the same cacheline at any given time instead of serializing the requests. 1 = URB Read CAM matching is disabled; multiple URB reads to the same cacheline are allowed to be concurrent (default). 0 = URB Read CAM matching is enabled; multiple URB reads to the same cacheline are serialized. lbcf_csr_lsqc_urbrdcam_dis.
	29:28	**Traffic regulation in LSQC for URB lookup traffic** Default Value: 00b Access: R/W Traffic regulation in LSQC for URB lookup traffic (URB lookups are issued to ltcc these many clocks apart). 00b - Continuous. 01b - 4 clocks apart. 10b - 8 clocks apart. 11b - 16 clocks apart. lbcf_lsqc_urb_traffic.
	27	**LQSC RO PERF DIS** Default Value: 0b Access: R/W Default: 0. when set, RO performance mode is disabled and all Reads proceed only after Parent recycles. lbcf_csr_lsqc_roperf_dis.

L3SQCREG4 - L3 SQC register 4

26 Order Cam Snp Reject

Default Value:	0b
Access:	R/W

Default: 0.
when set, all slots resulting in matches to snp addr result in snprsp as REJECT instead of MISS.
lbcf_csr_lsqc_ordercam_snpreject.

25 LQSC RW PERF DIS

Default Value:	0b
Access:	R/W

Default: 0.
0: Performance mode is enabled.
when set, Rd to RW performance mode is disabled and all cycles proceed only after Parent recycles.

lbcf_csr_lsqc_rwperf_dis.

24 LSQC read rtrn local crdt pre-consume disable

Default Value:	0b
Access:	R/W

0 - Default, LSQD consumes the LNE local slicecredit when read return pending.
1 - LSQD consumes read rtrn credit in the clock it is ready to send read return data.
lbcf_csr_lsqd_rdtrn_precrdt_dis.

23 LSQC Mem Write sqcam HITM response disable

Default Value:	0b
Access:	R/W

0 - Default.
1 - This disables any Memory Write from cache with HitM tag response to respond for SQCAMs.
lbcf_csr_lsqc_sqcam_l3tagrsphitm_dis.

22 Non-IA coherent atomics enable

Default Value:	1b
Access:	R/W

0: Atomics in GTI.
1: Atomics in L3 (non-IA atomic) (default).
lbcf_csr_lsqc_glblatmcs_l3.
Value of this bit should be same as LNCF register bit 0xb008[0].
Value of this bit should be same as LBCF register bit 0xb11c[8].

L3SQCREG4 - L3 SQC register 4

21	**Pipe line flush Coherent lines** Default Value: 0b Access: R/W 1: Treat pipeline flush as invalidating even coherent lines along with non coherent lines . 0: Flush invalidates non coherent lines only. lbcf_csr_lsqc_pipeflush_coh.
20:6	**Reserved** Access: RO
5	**Reserved2** Access: RO
4	**Reserved** Access: RO
3	**lslm flush denorm** Default Value: 0b Access: R/W lbcf_csr_lslm_flush_denorm_to_zero When this bit is enabled (1b), Floating Point SLM atomics output will be flushed to zero if it is a De-Norm.
2	**lsqc disable sla coh** Default Value: 0b Access: R/W lbcf_lsqc_disable_sla_coh If this bit is set to 1b, it will disable Short loop atomics access for Coherent atomics.
1	**lsqc disable sla** Default Value: 0b Access: R/W lbcf_lsqc_disable_sla If this bit is set to 1b, it will disable Short loop atomics access for Coherent and non-coherent atomics.
0	**lsqd flush denorm** Default Value: 0b Access: R/W lbcf_csr_lsqd_flush_denorm_to_zero When this bit is enabled (1b), Floating Point atomics output will be flushed to zero if it is a De-Norm.

L3 SQC registers 1

L3SQCREG1 - L3 SQC registers 1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00810000 [KBL]	
Size (in bits):	32	
Address:	0B100h	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO Reserved.
	23:19	**L3SQ General Priority Credit Initialization** Default Value: 10000b Access: R/W L3SQ General Priority Credit Initialization (SQGPCI): Number of general and high priority credits that SQ presents to L3 Arbiter blocks This inherently also determines the depth of the SQ; reduce the number of credits and SQ uses fewer slots. This field can be programmed only after a stalling flush Any value not listed here is considered Reserved. Gen priority credits is always greater than high priority credits. Value # General Credits 00000b 0 00001b 2 00010b 4 00011b 6 00100b 8 00101b 10 00110b 12 00111b 14 01000b 16 01001b

L3SQCREG1 - L3 SQC registers 1

		18 01010b 20 01011b 22 01100b 24 01101b 26 01110b 28 01111b 30 10000b 32 (default) ... 10100b 40 Need to go up to 40 credits. Ibcf_csr_lsqc_gen_credit_init[4:0].
	18:14	**L3SQ High Priority Credit Initialization** Default Value: 00100b Access: R/W L3SQ High Priority Credit Initialization (SQHPCI): Number of general and high priority credits that SQ presents to L3 Arbiter blocks. This inherently also determines the depth of the SQ; reduce the number of credits and SQ uses fewer slots. This field can be programmed only after a stalling flush Any value not listed here is considered Reserved. gen priority credits is always greater than high priority credits. Value # High Pri Credits 00000b 0 00001b 2 00010b 4 00011b 6 00100b 8 (default) 00101b 10 00110b

L3SQCREG1 - L3 SQC registers 1

		12 00111b 14 01000b 16 01001b 18 01010b 20 01011b 22 01100b 24 01101b 26 01110b 28 01111b 30 10000b 32 ... 10100b 40 Can to go up to 40 credits for SKLT lbcf_csr_lsqc_hp_credit_init[4:0] + lbcf_csr_lsqc_gen_credit_init[4:0] should always be less than or equal to 40(SKLT). lbcf_csr_lsqc_hp_credit_init[4:0].
	13:10	**Reserved** Access: RO Reserved.
	9	**L3SQ Read Once Enable for Sampler Client** Access: R/W L3SQ Read Once Enable for Sampler Client (SQROE): Enables Read Once indications to L3 Cache from SQ. Once enabled, any reads from Sampler client (MT) are sent as Read Once. 0 = (default) Reads from Sampler clients issue Read to L3 Cache. 1 = Reads from Sampler clients issue Read Once to L3 Cache. lbcf_csr_sampler_readonce_en.
	8:6	**Reserved** Access: RO Reserved.

L3SQCREG1 - L3 SQC registers 1

	5:3	**L3SQ Outstanding L3 Fills** Access: R/W L3SQ Outstanding L3 Fills (SQOUTSL3F): Identifies the number of L3 Fills that can be outstanding before SQ throttles the fill requests to L3 Cache. This is not an exact limit, but instead it is used as a threshold to throttling. Once the fill count is greater than or equal to the threshold, then no fills are issued until the fill responses are received to bring the outstanding count back below the threshold. 000b = (default) No limit. 001b = 1 fill. 010b = 2 fills. 011b = 4 fills. 100b = 8 fills. 101b = 16 fills. 11Xb = Reserved. lbcf_csr_lsqc_outs_fill[2:0].
	2:0	**L3SQ Outstanding L3 Lookups** Access: R/W L3SQ Outstanding L3 Lookups (SQOUTSL3L): Identifies the number of L3 lookups that can be outstanding before SQ throttles the lookup requests to L3 Cache. This is not an exact limit, but instead it is used as a threshold to throttling. once the lookup count is greater than or equal to the threshold, then no lookups are issued until the lookup responses are received to bring the outstanding count back below the threshold. 000b = (default) No limit. 001b = 1 lookup. 010b = 2 lookups. 011b = 4 lookups. 100b = 8 lookups. 101b = 16 lookups. 11Xb = Reserved. lbcf_csr_lsqc_outs_lookup[2:0].

L3 SQC registers 2

L3SQCREG2 - L3 SQC registers 2		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00004567	
Size (in bits):	32	
Address:	0B104h	
DWord	**Bit**	**Description**
0	31:17	**Reserved** Access: RO Reserved.
	16	**L3SQ Priority Selection Disable** Access: R/W L3SQ Priority Selection Disable (SQPRIDIS): Enables the use of priority selection based on client ID decodes. If disabled, all cycles in SQ are treated as same priority. 0 = (default) Priority selection is enabled. 1 = Priority selection is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority_cnt_disable.
	15	**L3SQ Priority 3 Pool Count Disable** Access: R/W L3SQ Priority 3 Pool Count Disable (SQPRI3CNTDIS): When set, priority3 pool becomes unlimited. And priority3 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 3 pool count is enabled. 1 = Priority 3 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority3_cnt_disable.
	14:12	**L3SQ Priority 3 Pool Counter** Default Value: 100b Access: R/W L3SQ Priority 3 Pool Counter (SQPRI3CNT): The count of cycles is selected from priority3 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request.

L3SQCREG2 - L3 SQC registers 2

		001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests. Ibcf_csr_priority3_cnt[2:0].
	11	**L3SQ Priority 2 Pool Count Disable** Access: R/W L3SQ Priority 2 Pool Count Disable (SQPRI2CNTDIS): When set, priority2 pool becomes unlimited. And priority2 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 2 pool count is enabled. 1 = Priority 2 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). Ibcf_csr_priority2_cnt_disable.
	10:8	**L3SQ Priority 2 Pool Counter** Default Value: 101b Access: R/W L3SQ Priority 2 Pool Counter (SQPRI2CNT): The count of cycles is selected from priority2 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests. Ibcf_csr_priority2_cnt[2:0].
	7	**L3SQ Priority 1 Pool Count Disable** Access: R/W L3SQ Priority 1 Pool Count Disable (SQPRI1CNTDIS): When set, priority1 pool becomes unlimited. And priority1 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 1 pool count is enabled. 1 = Priority 1 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). Ibcf_csr_priority1_cnt_disable.

L3SQCREG2 - L3 SQC registers 2

	6:4	**L3SQ Priority 1 Pool Counter** Default Value: 110b Access: R/W L3SQ Priority 1 Pool Counter (SQPRI1CNT): The count of cycles is selected from priority1 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests. Ibcf_csr_priority1_cnt[2:0].
	3	**L3SQ Priority 0 Pool Count Disable** Access: R/W L3SQ Priority 0 Pool Count Disable (SQPRI0CNTDIS): When set, priority0 pool becomes unlimited. And priority0 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 0 pool count is enabled. 1 = Priority 0 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). Ibcf_csr_priority0_cnt_disable.
	2:0	**L3SQ Priority 0 Pool Counter** Default Value: 111b Access: R/W L3SQ Priority 0 Pool Counter (SQPRI0CNT): The count of cycles is selected from priority0 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = (default) 128 requests. Ibcf_csr_priority0_cnt[2:0].

L3 SQC registers 3

L3SQCREG3 - L3 SQC registers 3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00001ABF
Size (in bits):	32
Address:	0B108h

DWord	Bit	Description
0	31:30	**Reserved** Access: RO Reserved.
	29:28	**SOLunit Priority Value** Access: R/W SOLunit Priority Value (SQSOLPRIVAL): Identifies the priority value for all cycles that are initiated by SOLunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_sol_priority[1:0].
	27:26	**GSunit Priority Value** Access: R/W GSunit Priority Value (SQGSPRIVAL): Identifies the priority value for all cycles that are initiated by GSunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_gs_priority[1:0].
	25:24	**TEunit Priority Value** Access: R/W TEunit Priority Value (SQTEPRIVAL): Identifies the priority value for all cycles that are initiated by TEunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default).

L3SQCREG3 - L3 SQC registers 3

		01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_te_priority[1:0].
	23:22	**CLunit Priority Value** Access: R/W CLunit Priority Value (SQCLPRIVAL): Identifies the priority value for all cycles that are initiated by CLunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_cl_priority[1:0].
	21:20	**TSunit Priority Value** Access: R/W TSunit Priority Value (SQTSPRIVAL): Identifies the priority value for all cycles that are initiated by TSunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_ts_priority[1:0].
	19:18	**SFunit Priority Value** Access: R/W SFunit Priority Value (SQSFPRIVAL): Identifies the priority value for all cycles that are initiated by SFunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_sf_priority[1:0].
	17:16	**SVSM Priority Value** Access: R/W SVSM Priority Value (SQSVSMPRIVAL): Identifies the priority value for all cycles that are initiated by SVSM. Priority is used in the L3 Super Queue (L3SQ).

L3SQCREG3 - L3 SQC registers 3

		00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_svsm_priority[1:0].
	15:14	**SARB Priority Value** Access: R/W SARB Priority Value (SQSARBPRIVAL): Identifies the priority value for all cycles that are initiated by State Arbiter (SARB). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. lbcf_csr_sarb_priority[1:0].
	13:12	**SBE Priority Value** Default Value: 01b Access: R/W SBE Priority Value (SQSBEPRIVAL): Identifies the priority value for all cycles that are initiated by SBE. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1 (default). 10b = Priority 2. 11b = Priority 3. lbcf_csr_sbe_priority[1:0].
	11:10	**IC$ Priority Value** Default Value: 10b Access: R/W IC$ Priority Value (SQICPRIVAL): Identifies the priority value for all cycles that are initiated by Instruction Cache (IC$). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. lbcf_csr_ic_priority[1:0].

L3SQCREG3 - L3 SQC registers 3

	9:8	**TDL Priority Value** Default Value: 10b Access: R/W TDL Priority Value (SQTDLPRIVAL): Identifies the priority value for all cycles that are initiated by TDL. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. lbcf_csr_tdl_priority[1:0].
	7:6	**DCunit Priority Value** Default Value: 10b Access: R/W DCunit Priority Value (SQDCPRIVAL): Identifies the priority value for all cycles that are initiated by DC. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. lbcf_csr_dc_priority[1:0].
	5:4	**DAPR Priority Value** Default Value: 11b Access: R/W DAPR Priority Value (SQDAPRPRIVAL): Identifies the priority value for all cycles that are initiated by DAPR. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_dapr_priority[1:0].
	3:2	**MTunit Priority Value** Default Value: 11b Access: R/W MTunit Priority Value (SQMTPRIVAL): Identifies the priority value for all cycles that are initiated by Sampler (MT). Priority is used in the L3 Super Queue (L3SQ).

L3SQCREG3 - L3 SQC registers 3

		00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_mt_priority[1:0].
	1:0	**LSQCunit Priority Value** Default Value: 11b Access: R/W LSQCunit Priority Value (SQPRIVAL): Identifies the priority value for all cycles that are initiated by Super Queue (L3 Evictions). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_lsqc_priority[1:0].

LBCF config save msg

LBCFCSR - LBCF config save msg		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B2FCh	
This register is not context saved and is written by PM unit.		
DWord	**Bit**	**Description**
0	31:10	**Reserved** Access: RO
	9:0	**Context save bit** Access: R/W Hardware Clear Bit[9]: Power Context Save Request. 0: Power context save is not being requested (default). 1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. Bits[8:0]: QWord Credits for Power Context Save Request. Minimum Credits = 1: Unit may send 1 QWord pair (enough for first LRI at least). Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and consumes one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

LBS config bits

LBSREG - LBS config bits

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x46000000 [KBL]
Size (in bits):	32
Address:	0B124h

Config Bits for LBS unit

DWord	Bit	Description
0	31:27	**Retry timer for lookup into LSQC** Default Value: 01000b Access: R/W Time between receiving Reject Response from LSQC and doing a snoop lookup request again onto LSQCunit. 00000b: 0 clocks. 00001b: 1 clocks. 00010b: 2 clocks. ... 01000b: 8 clocks (default value). ... 11111b: 32 clocks. lbcf_retry_timer[4:0].
	26	**Recycle parent faster in R/W perf mode** Default Value: 1b Access: R/W Arc into recycle as soon as parent becomes eligible to be recycled. 0: Disabled (recycle possible only when parent is recycled). 1: Enabled (default). lbcf_csr_lsqc_rwperf_quickrec.
	25	**Perf mode for Writes to same address** Default Value: 1b Access: R/W Performance improvement for writes to same address in L3: 0 - Performance mode is not enabled. 1 - Performance mode is enabled (default). lbcf_csr_lsqc_erlyrec.
	24:0	**Reserved** Access: RO

LCPLL1_CTL

LCPLL1_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	46010h-46013h
Name:	LCPLL 1 Control
ShortName:	LCPLL1_CTL
Power:	PG0
Reset:	global

The register is used to enable DPLL0 for driving the display core clock (CDCLK), the core display 2X clock (CD2XCLK), and the DDI ports.
DPLL frequency and port mapping programming is done through the DPLL_CTRL* registers.
This register is not reset by the device 2 FLR.

Restriction
Restriction : These fields must not be changed while any port or CDCLK is using DPLL0.

DWord	Bit	Description
0	31	**PLL Enable** This field enables or disables DPLL0. Value / Name: 0b Disable; 1b Enable Restriction : Configure DPLL0 frequency prior to enabling.
	30	**PLL Lock** Access: RO This read only bit indicates the status of the DPLL0 lock. Value / Name: 0b Not locked or not enabled; 1b Locked
	29:28	**Reserved**
	27:0	**Reserved** Format: MBZ

LCPLL2_CTL

LCPLL2_CTL

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	46014h-46017h
Name:	LCPLL 2 Control
ShortName:	LCPLL2_CTL
Power:	PG0
Reset:	soft

The register is used to enable DPLL1 for driving the DDI ports.
DPLL frequency, SSC, and port mapping programming is done through the DPLL_CTRL* and DPLL*_CFGCR* registers.

Restriction
Restriction : These fields must not be changed while any port is using DPLL1.

DWord	Bit	Description
0	31	**PLL Enable** This bit enables or disables DPLL1. Value: 0b – Disable; 1b – Enable Restriction : Configure DPLL1 frequency and SSC prior to enabling.
	30	**Reserved** Format: MBZ
	29:28	**Reserved**
	27:0	**Reserved** Format: MBZ

LINKM

LINKM	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	60040h-60043h
Name:	Transcoder A Link M Value 1
ShortName:	TRANS_LINKM1_A
Power:	PG2
Reset:	soft
Address:	61040h-61043h
Name:	Transcoder B Link M Value 1
ShortName:	TRANS_LINKM1_B
Power:	PG2
Reset:	soft
Address:	62040h-62043h
Name:	Transcoder C Link M Value 1
ShortName:	TRANS_LINKM1_C
Power:	PG2
Reset:	soft
Address:	6F040h-6F043h
Name:	Transcoder EDP Link M Value 1
ShortName:	TRANS_LINKM1_EDP
Power:	PG1
Reset:	soft

Description

There is one instance of this register for each transcoder A/B/C/EDP. This register is double buffered to update on the next MSA after LINKN is written.

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:0	**Link M value** This field is the link M value for external transmission in the Main Stream Attributes.

LINKN

LINKN	
Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	60044h-60047h
Name:	Transcoder A Link N Value 1
ShortName:	TRANS_LINKN1_A
Power:	PG2
Reset:	soft
Address:	61044h-61047h
Name:	Transcoder B Link N Value 1
ShortName:	TRANS_LINKN1_B
Power:	PG2
Reset:	soft
Address:	62044h-62047h
Name:	Transcoder C Link N Value 1
ShortName:	TRANS_LINKN1_C
Power:	PG2
Reset:	soft
Address:	6F044h-6F047h
Name:	Transcoder EDP Link N Value 1
ShortName:	TRANS_LINKN1_EDP
Power:	PG1
Reset:	soft

Description

There is one instance of this register for each transcoder A/B/C/EDP. This register is double buffered to update on the next MSA after written. **Writes to this register arm M/N registers for this transcoder.**

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:0	**Link N value** This field is the link N value for external transmission in the Main Stream Attributes and VB-ID.

LNCF config save msg

LNCFCSR - LNCF config save msg		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B0FCh	
This register is not context saved and is written by pm unit.		
DWord	**Bit**	**Description**
0	31:10	**Reserved** Access: RO
	9:0	**Context save bit** Access: R/W Hardware Clear Bit[9]. Power Context Save Request 0: Power context save is not being requested (default). 1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. Bits[8:0]. QWord Credits for Power Context Save Request. Minimum Credits = 1: Unit may send 1 QWord pair (enough for first LRI at least). Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

LNCF MOCS Register 0

LNCFCMOCS0 - LNCF MOCS Register 0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B020h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W [21:20] - L3 Cacheability Control (L3CC): Memory type information used in L3. This field is combined with the additional two bits that are sent by HDC based on binding table index. For all other L3 requesters, this field is the primary source of L3 cache controls. 00: Use binding table index for direct EU accesses - for rest it is reserved. 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB) [19:17] - Skip Caching Control (SCC) Defines the bit values to enable caching. Outcome overrides the L3/LLC caching for the surface. If "0", then corresponding address bit value is don't care Bit[1]=1: address bit[9] needs to be '0' to cache in target Bit[2]=1: address bit[10] needs to be '0' to cache in target Bit[3]=1: address bit[11] needs to be '0' to cache in target [16] - Enable Skip Caching (ESC) Enable for the Skip cache mechanism Skip caching needs to be disabled in SKLT 0: Not enabled 1: Enabled for L3 LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO

LNCFCMOCS0 - LNCF MOCS Register 0

	5:0	**MOCS lower data** Default Value: 000000b Access: R/W [5:4] - L3 Cacheability Control (L3CC): Memory type information used in L3. This field is combined with the additional two bits that are sent by HDC based on binding table index. For all other L3 requesters, this field is the primary source of L3 cache controls. 00: Use binding table index for direct EU accesses - for rest it is reserved. 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB) [3:1] - Skip Caching Control (SCC) Defines the bit values to enable caching. Outcome overrides the L3/LLC caching for the surface. If "0", then corresponding address bit value is don't care Bit[1]=1: address bit[9] needs to be '0' to cache in target Bit[2]=1: address bit[10] needs to be '0' to cache in target Bit[3]=1: address bit[11] needs to be '0' to cache in target [0] - Enable Skip Caching (ESC) Enable for the Skip cache mechanism Skip caching needs to be disabled in SKLT 0: Not enabled 1: Enabled for L3 LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 0

LNCFCMOCS0 - LNCF MOCS Register 0

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B020h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W [21:20] - L3 Cacheability Control (L3CC): Memory type information used in L3. This field is combined with the additional two bits that are sent by HDC based on binding table index. For all other L3 requesters, this field is the primary source of L3 cache controls. 00: Use binding table index for direct EU accesses - for rest it is reserved. 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB) [19:17] - Skip Caching Control (SCC) Defines the bit values to enable caching. Outcome overrides the L3/LLC caching for the surface. If "0", then corresponding address bit value is don't care Bit[1]=1: address bit[9] needs to be '0' to cache in target Bit[2]=1: address bit[10] needs to be '0' to cache in target Bit[3]=1: address bit[11] needs to be '0' to cache in target [16] - Enable Skip Caching (ESC) Enable for the Skip cache mechanism Skip caching needs to be disabled in SKLT 0: Not enabled 1: Enabled for L3 LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO

LNCFCMOCS0 - LNCF MOCS Register 0

	5:0	**MOCS lower data** Default Value: 000000b Access: R/W [5:4] - L3 Cacheability Control (L3CC): Memory type information used in L3. This field is combined with the additional two bits that are sent by HDC based on binding table index. For all other L3 requesters, this field is the primary source of L3 cache controls. 00: Use binding table index for direct EU accesses - for rest it is reserved. 01: Uncacheable (UC) - non-cacheable 10: Writethrough (WT) 11: Writeback (WB) [3:1] - Skip Caching Control (SCC) Defines the bit values to enable caching. Outcome overrides the L3/LLC caching for the surface. If "0", then corresponding address bit value is don't care Bit[1]=1: address bit[9] needs to be '0' to cache in target Bit[2]=1: address bit[10] needs to be '0' to cache in target Bit[3]=1: address bit[11] needs to be '0' to cache in target [0] - Enable Skip Caching (ESC) Enable for the Skip cache mechanism Skip caching needs to be disabled in SKLT 0: Not enabled 1: Enabled for L3 LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 1

LNCFCMOCS1 - LNCF MOCS Register 1

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B024h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 1

LNCFCMOCS1 - LNCF MOCS Register 1		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B024h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 2

LNCFCMOCS2 - LNCF MOCS Register 2

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B028h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 2

LNCFCMOCS2 - LNCF MOCS Register 2		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x0020001F	
Size (in bits):	32	
Address:	0B028h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 3

LNCFCMOCS3 - LNCF MOCS Register 3

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B02Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 3

LNCFCMOCS3 - LNCF MOCS Register 3		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B02Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 4

LNCFCMOCS4 - LNCF MOCS Register 4

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B030h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 4

LNCFCMOCS4 - LNCF MOCS Register 4		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x0030001F	
Size (in bits):	32	
Address:	0B030h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 5

LNCFCMOCS5 - LNCF MOCS Register 5		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B034h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 5

LNCFCMOCS5 - LNCF MOCS Register 5

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B034h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 6

LNCFCMOCS6 - LNCF MOCS Register 6

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B038h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 6

LNCFCMOCS6 - LNCF MOCS Register 6

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B038h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 7

LNCFCMOCS7 - LNCF MOCS Register 7

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B03Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 7

LNCFCMOCS7 - LNCF MOCS Register 7		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B03Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 8

LNCFCMOCS8 - LNCF MOCS Register 8

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B040h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 8

LNCFCMOCS8 - LNCF MOCS Register 8		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00100000	
Size (in bits):	32	
Address:	0B040h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 9

LNCFCMOCS9 - LNCF MOCS Register 9

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B044h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 9

LNCFCMOCS9 - LNCF MOCS Register 9		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B044h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 10

LNCFCMOCS10 - LNCF MOCS Register 10

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B048h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 10

LNCFCMOCS10 - LNCF MOCS Register 10		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x0020001F	
Size (in bits):	32	
Address:	0B048h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 11

LNCFCMOCS11 - LNCF MOCS Register 11

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B04Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 11

LNCFCMOCS11 - LNCF MOCS Register 11		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B04Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 12

LNCFCMOCS12 - LNCF MOCS Register 12

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B050h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 12

LNCFCMOCS12 - LNCF MOCS Register 12

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B050h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 13

LNCFCMOCS13 - LNCF MOCS Register 13

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B054h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 13

LNCFCMOCS13 - LNCF MOCS Register 13		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B054h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 14

LNCFCMOCS14 - LNCF MOCS Register 14

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B058h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 14

LNCFCMOCS14 - LNCF MOCS Register 14

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B058h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 15

LNCFCMOCS15 - LNCF MOCS Register 15

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B05Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 15

LNCFCMOCS15 - LNCF MOCS Register 15

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B05Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 16

LNCFCMOCS16 - LNCF MOCS Register 16

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B060h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 16

LNCFCMOCS16 - LNCF MOCS Register 16

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B060h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 17

LNCFCMOCS17 - LNCF MOCS Register 17

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B064h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 17

LNCFCMOCS17 - LNCF MOCS Register 17		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B064h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 18

LNCFCMOCS18 - LNCF MOCS Register 18

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B068h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 18

LNCFCMOCS18 - LNCF MOCS Register 18

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B068h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 19

LNCFCMOCS19 - LNCF MOCS Register 19

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B06Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 19

LNCFCMOCS19 - LNCF MOCS Register 19

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B06Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 20

LNCFCMOCS20 - LNCF MOCS Register 20

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B070h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 20

LNCFCMOCS20 - LNCF MOCS Register 20

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B070h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 21

LNCFCMOCS21 - LNCF MOCS Register 21

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B074h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 21

LNCFCMOCS21 - LNCF MOCS Register 21		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B074h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 22

LNCFCMOCS22 - LNCF MOCS Register 22

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B078h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 22

LNCFCMOCS22 - LNCF MOCS Register 22

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B078h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 23

LNCFCMOCS23 - LNCF MOCS Register 23

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B07Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 23

LNCFCMOCS23 - LNCF MOCS Register 23

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B07Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 24

LNCFCMOCS24 - LNCF MOCS Register 24

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B080h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 24

LNCFCMOCS24 - LNCF MOCS Register 24

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00100000
Size (in bits):	32
Address:	0B080h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 25

LNCFCMOCS25 - LNCF MOCS Register 25

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B084h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 25

LNCFCMOCS25 - LNCF MOCS Register 25

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B084h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 26

LNCFCMOCS26 - LNCF MOCS Register 26

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B088h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 26

LNCFCMOCS26 - LNCF MOCS Register 26

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0020001F
Size (in bits):	32
Address:	0B088h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 100000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 27

LNCFCMOCS27 - LNCF MOCS Register 27		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B08Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 27

LNCFCMOCS27 - LNCF MOCS Register 27		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B08Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 28

LNCFCMOCS28 - LNCF MOCS Register 28

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B090h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 28

LNCFCMOCS28 - LNCF MOCS Register 28

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0030001F
Size (in bits):	32
Address:	0B090h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 110000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 29

LNCFCMOCS29 - LNCF MOCS Register 29

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00170013
Size (in bits):	32
Address:	0B094h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 29

LNCFCMOCS29 - LNCF MOCS Register 29		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00170013	
Size (in bits):	32	
Address:	0B094h	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 010111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 010011b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 30

LNCFCMOCS30 - LNCF MOCS Register 30

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B098h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 30

LNCFCMOCS30 - LNCF MOCS Register 30

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x0000001F
Size (in bits):	32
Address:	0B098h

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 011111b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 31

LNCFCMOCS31 - LNCF MOCS Register 31

Register Space:	MMIO: 0/2/0
Source:	BSpec
Default Value:	0x00000000
Size (in bits):	32
Address:	0B09Ch

This register is for Mocs index

DWord	Bit	Description
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF MOCS Register 31

LNCFCMOCS31 - LNCF MOCS Register 31		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B09Ch	
This register is for Mocs index		
DWord	**Bit**	**Description**
0	31:22	**Reserved** Access: RO
	21:16	**MOCS upper data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.
	15:6	**Reserved** Access: RO
	5:0	**MOCS lower data** Default Value: 000000b Access: R/W Refer LNCF MOCS Register 0 for detailed description LNCF MOCS implementation does not support write to individual MOCS index. S/W needs to write both upper and lower index together. If S/W needs to write to a single index, it should read,merge and write it back.

LNCF Render config save msg

LNCFRCSR - LNCF Render config save msg		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B0F8h	
This register is not context saved and is written by CS unit		
DWord	**Bit**	**Description**
0	31:17	**Reserved** Access: RO
	16	**Render Context save Request Mask** Access: R/W Hardware Clear Bit[16] Render Context Save Request Mask 0 : LNCFRCSR.Bit[0] is masked (default) 1 : LNCFRCSR.Bit[0] is valid
	15:1	**Reserved** Access: RO
	0	**Render Context save Request** Access: R/W Hardware Clear Bit[0] Render Context Save Request 0 : Render context save is not being requested (default) 1 : Render context save is being requested Unit needs to self-clear this bit upon sampling.

Load Indirect Base Vertex

3DPRIM_BASE_VERTEX - Load Indirect Base Vertex	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02440h-02443h

DWord	Bit	Description
0	31:0	**Base Vertex** Format: S31 This register is used to store the Base Vertex of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Instance Count

3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02438h-0243Bh

DWord	Bit	Description
0	31:0	**Instance Count** This register is used to store the Instance Count of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Start Instance

3DPRIM_START_INSTANCE - Load Indirect Start Instance	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0243Ch-0243Fh

DWord	Bit	Description
0	31:0	**Start Vertex** Format: U32 This register is used to store the Start Instance of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Start Vertex

3DPRIM_START_VERTEX - Load Indirect Start Vertex	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02430h-02433h

DWord	Bit	Description
0	31:0	**Start Vertex** Format: U32 This register is used to store the Start Vertex of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Vertex Count

3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count	
Register Space:	MMIO: 0/2/0
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02434h-02437h

DWord	Bit	Description
0	31:0	**Vertex Count** Format: U32 This register is used to store the Vertex Count of the 3D_PRIMITIVE command when Load Indirect Enable is set.

LPFC control register

LPFCCNTL - LPFC control register		
Register Space:	MMIO: 0/2/0	
Source:	BSpec	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B01Ch	
LPFC control register.		
DWord	**Bit**	**Description**
0	31	**LPFC enable signal** Access: R/W LPPC event collection enable signal. Incf_lpfc_cnt_en.
	30	**LPFC Global enable signal** Access: R/W LPFCSL unit global enable signal. 0b - LPFCSL unit is disabled. 1b - LPFCSL unit is enabled. Incf_lpfc_gbl_en
	29:0	**Reserved** Access: RO Reserved.

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 12: Display

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

VGA and Extended VGA Registers 1

General Control and Status Registers 2

ST00 - Input Status 0 3

ST01 - Input Status 1 4

FCR - Feature Control 6

MSR - Miscellaneous Output 7

Sequencer Registers 9

SRX - Sequencer Index 9

SR00 - Sequencer Reset 9

SR01 - Clocking Mode 10

SR02 - Plane/Map Mask 11

SR03 - Character Font 12

SR04 - Memory Mode Register 13

SR07 - Horizontal Character Counter Reset 14

Graphics Controller Registers 14

GRX - GRX Graphics Controller Index Register 15

GR00 - Set/Reset Register 15

GR01 - Enable Set/Reset Register 16

GR02 - Color Compare Register 16

GR03 - Data Rotate Register 17

GR04 - Read Plane Select Register 17

GR05 - Graphics Mode Register 18

GR06 - Miscellaneous Register 22

GR07 - Color Don't Care Register 23

GR08 - Bit Mask Register 24

GR10 - Address Mapping 24

GR11 - Page Selector 26

GR18 - Software Flags 26

Attribute Controller Registers 27

ARX - Attribute Controller Index Register 28

AR[00:0F] - Palette Registers [0:F] 28

AR10 - Mode Control Register 29
AR11 - Overscan Color Register 31
AR12 - Memory Plane Enable Register 32
AR13 - Horizontal Pixel Panning Register 33
AR14 - Color Select Register 34
VGA Color Palette Registers 35
DACMASK - Pixel Data Mask Register 36
DACSTATE - DAC State Register 36
DACRX - Palette Read Index Register 37
DACWX - Palette Write Index Register 37
DACDATA - Palette Data Register 38
CRT Controller Register 39
CRX - CRT Controller Index Register 40
CR00 - Horizontal Total Register 40
CR01 - Horizontal Display Enable End Register 41
CR02 - Horizontal Blanking Start Register 41
CR03 - Horizontal Blanking End Register 42
CR04 - Horizontal Sync Start Register 43
CR05 - Horizontal Sync End Register 44
CR06 - Vertical Total Register 45
CR07 - Overflow Register (Vertical) 46
CR08 - Preset Row Scan Register 49
CR09 - Maximum Scan Line Register 50
CR0A - Text Cursor Start Register 51
CR0B - Text Cursor End Register 52
CR0C - Start Address High Register 53
CR0D - Start Address Low Register 53
CR0E - Text Cursor Location High Register 54
CR0F - Text Cursor Location Low Register 54
CR10 - Vertical Sync Start Register 55
CR11 - Vertical Sync End Register 56
CR12 - Vertical Display Enable End Register 57
CR13 - Offset Register 57

CR14 - Underline Location Register 58
CR15 - Vertical Blanking Start Register 59
CR16 - Vertical Blanking End Register 59
CR17 - CRT Mode Control 60
CR18 - Line Compare Register 64
CR22 - Memory Read Latch Data Register 65
CR24 - Toggle State of Attribute Controller Register 65
Display Audio Codec Verbs 66
Block Diagram 66
Codec Node Hierarchy 67
Programming 68
Verb Support 68
Parameter Support 70
Node ID 00h Root Node Verbs 71
F00h - Get Parameters 71
Parameter 00h: VID - Vendor ID 71
Parameter 02h: RID - Revision ID 71
Parameter 04h: PARAM_SNC - Subordinate Node Count 71
F37h GET CCF - Get Current Clock Frequency 72
Node ID 01h Audio Function Group Verbs 72
F00h Get Parameters 72
Parameter 04h: PARAM_SNC - Subordinate Node Count 73
Parameter 05h: PARAM_FGT - Function Group Type 73
Parameter 08h: PARAM_FGC - Function Group Capability 73
Parameter 0Fh: PARAM_SPS - Supported Power States 73
Parameter 16h: PARAM_A2CAP - Azalia 2 Capabilities 74
705h SET_PS - Set Power State 74
F05h GET_PS - Get Power State 74
F20h GET SSID - Get Subsystem ID0 75
720h SET SSID0 - Set Subsystem ID0 75
721h SET SSID1 - Set Subsystem ID1 75
722h SET SSID2 - Set Subsystem ID2 75
723h SET SSID3 - Set Subsystem ID3 75

724h SET CCF - Set Current Clock Frequency....75
F24h GET CCF - Get Current Clock Frequency....76
7FFh SET Function Group Reset....76
Node ID 02h 03h 04h Audio Output Convertor Widget Verbs....76
2hAh SETGET_SDF - SetGET Stream Descriptor Format....77
F00h Get Parameters....77
Parameter 09h: AWC - Audio Widget Capabilities....78
Parameter 0Ah: PSB - PCM Sizes and Bit Rates....79
Parameter 0Bh: SF - Stream Formats....80
Parameter 0Fh: PARAM_SPS - Supported Power States....80
705h SET_PS - Set Power State....80
F05h GET_PS - Get Power State....80
706hF06h GETSET_CSID - GetSet Channel and Stream ID....81
Digital Converter Verbs....81
70Dh: SET_DC1 - Set Digital Converter 1....82
70Eh: Digital Converter 2....82
73Eh: Digital Converter 3....82
73Fh: Digital Converter 4....83
72DhF2Dh GETSET_CCC - GetSet Converter Channel Count....83
Node ID 05h 06h 07h Pin Widget Verbs....83
3h SET_AM - Set Amplifier Mute....84
B8h GET_AM - Get Amplifier Mute....84
F00h Get Parameters....84
Parameter 09h: AWC - Audio Widget Capabilities....85
Parameter 0Ch: PC - Pin Capabilities....85
Parameter 0Eh: CLL - Connection List Length....86
Parameter 12h: OAC - Output Amplifier Capabilities....86
Parameter 15h: DLL - Device List Length....87
Parameter 0Fh: PARAM_SPS - Supported Power States....87
701hF01h SETGET_CSC - SetGet Connection Select Control....87
F02h GET_CLE - Get Connection List Entry....87
705h SET_PS - Set Power State....88
F05h GET_PS - Get Power State....88

707hF07h SETGET_PWC - SetGet Pin Widget Control 88
708hF08h SETGET_UE - SetGet Unsolicited Enable 88
F09h GET_PS - Get Pin Sense 89
71Ch SET_CD0 - Set Configuration Default Byte 0 89
71Dh SET_CD1 - Set Configuration Default Byte 1 89
71Eh SET_CD2 - Set Configuration Default Byte 2 89
71Fh SET_CD3 - Set Configuration Default Byte 3 89
F1Ch GET_CD - Get Configuration Default 90
F2Eh HDMIDP Info Size 90
F2Fh Get ELD Data 91
Parameter nn: ELD Data 91
730hF30h SETGET_HII - SetGet HDMI Info Index 91
731hF31h SETGET_HID - SetGet HDMI Info Data 91
732hF32h SETGET_HITC - SetGet HDMI Info Transmit Control 91
733h SET_PC - Set Protection Control 92
734hF34h SETGET_CCM - GetSet Converter Channel Map 92
735h SET_DS - Set Device Select 92
F35h: GET_DS - Get Device Select 92
F36h GET_DDLE - Get Display Device List Entry 92
73ChF3Ch SETGET_DPID - SetGet DisplayPort Stream ID 93
Node ID 08h Intel Vendor Widget Verbs 93
F00h Get Parameters 94
Parameter 09h: AWC - Audio Widget Capabilities 94
71Eh SET_GET_GFXMAILBOX - Set Get GFX MAILBOX Byte 2 94
728h SET CLOCK OFF - Set Clock Off Command 94
708hF08h SETGET_UE - SetGet Unsolicited Enable 94
781hF81h GETSET_VV - GetSet iDisp Codec Vendor Verb 95
782h SET_GTCT - Set GTC Trigger 95
F83h GET_CGTC - Get Captured GTC Value 95
F84h GET_CWC - Get Captured Wall Clock Value 95
F85h GET GOF - Get GTC Offset Value 96
785h SET GOF0 - Set GTC Offset Value Byte 0 96
786h SET GOF1 - Set GTC Offset Value Byte 1 96

787h SET GOF2 - Set GTC Offset Value Byte 2 96
788h SET GOF3 - GTC Offset Value Byte 3 96
789hF89h SETGET_GDI - SetGet GTC Device Index 96
3h SET_AM - Set Amplifier Mute 96
B8h GET_AM - Get Amplifier Mute 97
F00h Get Parameters 97
Parameter 09h: AWC - Audio Widget Capabilities 97
Parameter 0Ch: PC - Pin Capabilities 98
Parameter 0Eh: CLL - Connection List Length 99
Parameter 12h: OAC - Output Amplifier Capabilities 99
Parameter 0Fh: PARAM_SPS - Supported Power States 99
701hF01h SETGET_CSC - SetGet Connection Select Control 100
F02h GET_CLE - Get Connection List Entry 100
705h SET_PS - Set Power State 100
F05h GET_PS - Get Power State 100
707hF07h SETGET_PWC - SetGet Pin Widget Control 101
708hF08h SETGET_UE - SetGet Unsolicited Enable 101
F09h GET_PS - Get Pin Sense 101
71Ch SET_CD0 - Set Configuration Default Byte 0 102
71Dh SET_CD1 - Set Configuration Default Byte 1 102
71Eh SET_CD2 - Set Configuration Default Byte 2 102
71Fh SET_CD3 - Set Configuration Default Byte 3 102
F1Ch GET_CD - Get Configuration Default 103
F2Fh Get ELD Data 103
Parameter nn: ELD Data 103
733h SET_PC - Set Protection Control 103
734hF34h SETGET_CCM - GetSet Converter Channel Map 104
740hF40h SETGET PTS Offset Byte0 104
741hF41h SETGET PTS Offset Byte1 104
742hF42h SETGET PTS Offset Byte2 104
743hF43h SETGET PTS Offset Byte3 104
North Display Engine Registers 105
Display Connections 105

Display Pipes 106
Display Transcoders 107
Audio 107
DDIs 107
Pipe to Transcoder to DDI Mappings 109
Terminology 110
Mode Set 112
Sequences to Initialize Display 112
Initialize Sequence 112
Un-initialize Sequence 113
Sequences for DisplayPort 113
Enable Sequence 113
Notes 114
Enabling DisplayPort Sync Mode 115
Disable Sequence 115
Disabling DisplayPort Sync Mode 116
Sequences for HDMI and DVI 117
Enable Sequence 117
Notes 117
Disable Sequence 118
Sequences for Display C5 and C6 119
Sequence to Allow DC5 or DC6 119
Sequence to Disallow DC5 and DC6 120
DMC Firmware Package 120
CSS Header 122
Package Header 122
DMC firmware binary 123
Display Resolution Support 124
Maximum Pipe Pixel Rate 124
Maximum Port Link Rate 125
Maximum Memory Read Bandwidth 125
Maximum Watermark 126
Display Resolution Capabilities 126

Examples 127
Clocks 128
Overview of Supported Display Clock Paths 129
Display Engine Clock Reference 130
Display Engine PLLs 130
Recommended PLL Selection 131
DDI Clocks 131
Transcoder Clocks 131
CD Clock 132
Port Clock Programming 133
DisplayPort Programming 133
DisplayPort PLL Enable Sequence 133
DisplayPort PLL Disable Sequence 133
Example of DisplayPort on DDIA using HBR 2.7 GHz link rate with SSC 133
HDMI and DVI Programming 134
HDMI and DVI PLL Enable Sequence 134
HDMI and DVI PLL Disable Sequence 134
Formula for HDMI and DVI DPLL Programming 134
Algorithm to Find HDMI and DVI DPLL Programming 134
Pseudo-code for HDMI and DVI DPLL Programming 135
Example of DVI on DDIB using 113.309 MHz symbol clock 136
Example of HDMI on DDIC using 296.703 MHz symbol clock 137
Sequences for Changing CD Clock Frequency 138
Resets 139
Shared Functions 140
Fuses and Straps 140
Interrupts 140
Interrupt Flow 141
Interrupt Service Routine 142
Render Response 143
Arbiter 143
GSA 143
Data Buffer 143

Backlight 144
Backlight Enabling Sequence 144
Backlight Registers 145
Miscellaneous Shared Functions 145
Central Power 145
Frame Buffer Compression 145
FBC Registers 145
FBC Overview 145
FBC Compression Limit 146
FBC Programming Overview 146
Render Tracking With Nuke 147
Render Tracking Without Nuke 147
Blitter Tracking With Nuke 148
Blitter Tracking Without Nuke 148
CPU Host Aperture Tracking 149
Display Plane Enabling with FBC 149
Display Plane Disabling with FBC 149
Watermarks 150
DC States 150
Power Wells 150
Pipe 150
Color Space Conversion 150
Pipe Color Gamut Enhancement 153
Pipe DPST 155
Pipe Palette and Gamma 155
Programming Modes 155
8 bit legacy palette/gamma mode: 155
10 bit gamma mode: 156
Split gamma mode: 156
12 bit interpolated gamma mode: 157
Example Pipe Gamma Correction Curve 158
Pipe Control 159
Pipe Scaler 159

Planes 160
Plane Capability and Interoperability 160
Plane Assignments and Capabilities 160
Plane Feature Interoperability 161
Render/Display Decompression 162
Color Control Surface 162
Decompression Programming 163
Plane Rotation Programming 163
90 Rotation 163
270 rotation 167
Display Buffer Programming 167
Display Buffer Allocation 167
Display Buffer Size 167
Allocation Requirements 167
Minimum Allocation Requirements 168
Basic Allocation Method 168
Single Pipe 168
Multi-Pipe 169
Buffer allocation re-distribution 170
Display Buffer Allocation and Watermark programming prior to OS boot 170
VGA 171
Cursor Plane 171
Universal Plane 171
Plane Pixel Formats 173
Transcoder 174
Transcoder Control 174
Transcoder Timing 174
Transcoder MN Values 174
Transcoder Video Data Island Packet 176
Transcoder DDI Function 179
Panel Self Refresh 179
Transcoder Port Sync 180
Feature Description 180

DP/eDP Port Sync Restrictions 180
Audio 180
Audio Bios Programming Sequence 180
Codec Verb Table 180
Audio Programming Sequence for Link wakeup 183
Audio Programming Sequence 184
Audio Configuration 186
Digital Display Interface 186
DDI Buffer 186
I_boost 187
Recommended Buffer Translation Programming 187
DDI AUX Channel 190
AUX programming sequence 191
DisplayPort Transport 192
Global Time Code (GTC) 192
Top Level GTC 192
DDI Level GTC 192
South Display Engine Registers 193
Terminology 194
Shared Functions 195
Fuses and Straps 195
Raw Clock 195
Interrupts and Hot Plug 195
Panel Power and Backlight 196
Panel Power 196
Backlight 196
Backlight Enabling Sequence 197
Backlight Registers 197
GMBUS and GPIO 198
Registers 198
Pin Usage 198
GPIO Programming for I2C Bit Bashing 199
GMBUS Controller Programming Interface 199

Sequence for GMBUS Burst Reads Greater Than 511 Bytes 199
Display Watermark Programming 200
Watermark Overview 200
Watermark Calculations 200
Watermark Algorithm 201
Transition Watermark 204
Scaling 205
System Agent Geyserville (SAGV) 206
Sequence to Disable SAGV 206
Sequence to Enable SAGV 206
Examples 207
Example pixel rate adjustments: 207
Example method, block, and line calculations: 207
Memory Values 208
Retrieve Memory Latency Data 208
Memory Latency Data Definition 208
SAGV Block Time 209

VGA and Extended VGA Registers

This section describes the registers and the functional operation notations for the observable registers in the VGA section. This functionality is provided as a means for support of legacy applications and operating systems. It is important to note that these registers in general have the desired effects only when running VGA display modes.

The main exceptions to this are the palette interface which allows real mode DOS applications and full screen VGA applications under an OS control running in high resolution (non-VGA) modes to access the palette through the VGA register mechanisms and the use of the ST01 status bits that determine when the VGA enters display enable and sync periods. Other exceptions include the register bits that control the memory accesses through the A000:0000 and B000:0000 memory segments which are used during operating system emulation of VGA for "DOS box" applications. Some of the functions of the VGA are enabled or defeated through the programming of the VGA control register bits that are located in the MMIO register space.

Given the legacy nature of this function, it has been adapted to the changing environment that it must operate within. The three most notable changes are the addition of high resolution display mode support, new operating system support, and the use of fixed resolution display devices (such as LCD panels). Additional control bits in the PCI Config space will affect the ability to access the registers and memory aperture associated with VGA.

Mode of Operation	VGA Disable	VGA Display	VGA Registers	Palette (VGA)	VGA Memory	VGA Banking
VGA DOS	No	Yes	Yes	Yes	Yes	No
HiRes DOS	Yes	No	Yes	Yes	No	Yes
Fullscreen DOS	Yes/No	No/Yes	Yes	Yes	Yes	Yes
DOS Emulation	Yes	No	Yes	Yes	Yes	Yes

VGA Display Mode	Dot Clock Select	Dot Clock Range	132 Column Text Support	9-Dot Disable Support	Main Use
Native	VGA Clock Select	25/28 MHz	No	No	Analog CRT (VGA connector)
Centered	Fixed at display Requirements	Product Specific	No	Yes	Digital Display
Upper Left Corner	Fixed at display Requirements	Product Specific	No	Yes	Internal Panel

Native, Centered, and Upper Left Corner support varies from product to product.

Even in the native VGA display operational modes, not all combinations of bit settings result in functional operating modes. VGA display modes have the restriction that they can be used only when all other display planes are disabled.

These registers are accessed via I/O space. The I/O space resides in the PCI compatibility hole and uses only the addresses that were part of the original VGA I/O space (which includes EGA and MDA emulation). Accesses to the VGA I/O addresses are steered to the proper bus and rely on proper setup of bridge registers. Extended VGA registers such as GR10 and GR11 use additional indexes for the already defined I/O addresses. VGA register accesses are allowed as 8 or 16 bit naturally aligned transactions only. Word transactions must have the least significant bit of the address set to zero. DWORD I/O operations should not be performed on these registers.

Some products may support access to these registers through MMIO. The access method varies and is documented elsewhere.

General Control and Status Registers

The setup, enable, and general registers are all directly accessible by the CPU. A sub indexing scheme is not used to read from and write to these registers.

Name	Function	Read		Write	
		I/O	Memory Offset	I/O	Memory Offset
ST00	VGA Input Status Register 0	3C2h	3C2h	--	--
ST01	VGA Input Status Register 1	3BAh/3DAh[1]	3BAh/3DAh[1]	--	--
FCR	VGA Feature Control Register	3CAh	3CAh	3BAh/3DAh[1]	3BAh/3DAh[1]
MSR	VGA Miscellaneous Output Register	3CCh	3CCh	3C2h	3C2h

[1] The address selection for ST01 reads and FCR writes is dependent on CGA or MDA emulation mode as selected via the MSR register.

Various bits in these registers provide control over the real-time status of the horizontal sync signal, the horizontal retrace interval, the vertical sync signal, and the vertical retrace interval. The horizontal retrace interval is the period during the drawing of each scan line containing active video data, when the active video data is not being displayed. This period includes the horizontal front and back porches, and the horizontal sync pulse. The horizontal retrace interval is always longer than the horizontal sync pulse. The vertical retrace interval is the period during which the scan lines not containing active video data are drawn. This includes the vertical front porch, back porch, and the vertical sync pulse. The vertical retrace interval is normally longer than the vertical sync pulse.

ST00 - Input Status 0

I/O (and Memory Offset) Address: 3C2h

Default: 00h

Attributes: Read Only

Bit	Descriptions
7	**CRT Interrupt Pending.** This bit is here for EGA compatibility and **will always return zero**. The generation of interrupts was originally enabled, through bits [4,5] of the Vertical Retrace End Register (CR11). This ability to generate interrupts at the start of the vertical retrace interval is a feature that is typically unused by DOS software and therefore is only supported through other means for use under a operating system support. 0 = CRT (vertical retrace interval) interrupt is not pending. 1 = CRT (vertical retrace interval) interrupt is pending
6:5	**Reserved.** Read as 0s.
4	**RGB Comparator / Sense.** This bit is here for compatibility and **will always return one**. Monitor detection must be done be done through the programming of registers in the MMIO space. 0 = Below threshold 1 = Above threshold
3:0	**Reserved.** Read as 0s.

ST01 - Input Status 1

I/O (and Memory Offset) Address: 3BAh/3DAh

Default: 00h

Attributes: Read Only

The address selection is dependent on CGA or MDA emulation mode as selected via the MSR register.

Bit	Descriptions
7	**Reserved (as per VGA specification).** Read as 0s.
6	**Reserved.** Read as 0.
5:4	**Video Feedback 1, 0.** These bits are connected to 2 of the 8 color bits sent to the palette. Bits 4 and 5 of the Color Plane Enable Register (AR12) selects which two of the 8 possible color bits become connected to these 2 bits of this register. These bits exist for EGA compatibility.
3	**Vertical Retrace/Video.** 0 = VSYNC inactive (Indicates that a vertical retrace interval is not taking place). 1 = VSYNC active (Indicates that a vertical retrace interval is taking place). VGA pixel generation is not locked to the display output but is loosely coupled. A VSYNC indication may not occur during the actual VSYNC going to the display but during the VSYNC that is generated as part of the VGA pixel generation. The exact relationship will vary with the VGA display operational mode. This status bit will remain active when the VGA is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now incorrectly) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. Bits 4 and 5 of the Vertical Retrace End Register (CR11) previously could program this bit to generate an interrupt at the start of the vertical retrace interval. This ability to generate interrupts at the start of the vertical retrace interval is a feature that is largely unused by legacy software. Interrupts are not supported through the VGA register bits.
2:1	**Reserved.** Read as 0s.

Bit	Descriptions
0	**Display Enable Output.** Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. This bit was used with the EGA graphics system (and the ones that preceded it, including MDA and CGA). In those cases, it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to the frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer. Those behaviors resulted in either "snow" or a flickering display. This bit provides compatibility with software designed for those early graphics controllers. This bit is currently used in DOS applications that access the palette to prevent the sparkle associated with read and write accesses to the palette RAM with the same address on the same clock cycle. **This status bit remains active when the VGA display is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now considered incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. When in panel fitting VGA or centered VGA operation, the meaning of these bits will not be consistent with native VGA timings.** 0 = Active display data is being sent to the display. Neither a horizontal retrace interval or a vertical retrace interval is currently taking place. 1 = Either a horizontal retrace interval (horizontal blanking) or a vertical retrace interval (vertical blanking) is currently taking place.

FCR - Feature Control

I/O (and Memory Offset) Address: 3BAh/3DAh - Write; 3CAh - Read

Default: 00h

Attributes: See Address above

The I/O address used for writes is dependent on CGA or MDA emulation mode as selected via the MSR register. In the original EGA, bits 0 and 1 were used as part of the feature connector interface. Feature connector is not supported in these devices and those bits will always read as zero.

Bit	Descriptions
7:4	**Reserved.** Read as 0.
3	**VSYNC Control.** This bit is provided for compatibility only and has no other function. Reads and writes to this bit have no effect other than to change the value of this bit. The previous definition of this bit selected the output on the VSYNC pin. 0 = Was used to set VSYNC output on the VSYNC pin (default). 1 = Was used to set the logical 'OR' of VSYNC and Display Enable output on the VSYNC pin. This capability was not typically very useful.
2:0	**Reserved.** Read as 0.

MSR - Miscellaneous Output

I/O (and Memory Offset) Address: 3C2h - Write; 3CCh - Read

Default: 00h

Attributes: See Address above

Bit	Descriptions
7	**CRT VSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. Sync polarity was used in VGA to signal the monitor how many lines of active display are being generated. 0 = Positive Polarity (default). 1 = Negative Polarity.
6	**CRT HSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. 0 = Positive Polarity (default). 1 = Negative Polarity
5	**Page Select.** In Odd/Even Memory Map Mode 1 (GR6), this bit selects the upper or lower 64 KB page in display memory for CPU access: 0 = Upper page (default) 1 = Lower page. Selects between two 64KB pages of frame buffer memory during standard VGA odd/even modes (modes 0h through 5h). Bit 1 of register GR06 can also program this bit in other modes. This bit is would normally set to 1 by the software.
4	**Reserved.** Read as 0.
3:2	**Clock Select.** These bits can select the dot clock source for the CRT interface. The bits should be used to select the dot clock in standard native VGA modes only. When in the centering or upper left corner modes, these bits should be set to have no effect on the clock rate. The actual frequencies that these bits select, if they have any affect at all, is programmable through the DPLL MMIO registers. 00 = CLK0, 25.175 MHz (for standard VGA modes with 640 pixel (8-dot) horizontal resolution) (default) 01 = CLK1, 28.322 MHz. (for standard VGA modes with 720 pixel (9-dot) horizontal resolution) 10 = Was used to select an external clock (now unused) 11 = Reserved

Bit	Descriptions
1	**A0000-BFFFFh Memory Access Enable.** VGA Compatibility bit enables access to video memory (frame buffer) at A0000-BFFFFh. When disabled, accesses to VGA memory are blocked in this region. This bit is independent of and does not block CPU access to the video linear frame buffer at other addresses. 0 = Prevent CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture (default). 1 = Allow CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture. This memory must be mapped as UC by the CPU.
0	**I/O Address Select.** This bit selects 3Bxh or 3Dxh as the I/O address for the CRT Controller registers, the Feature Control Register (FCR), and Input Status Register 1 (ST01). Presently ignored (whole range is claimed), but will "ignore" 3Bx for color configuration or 3Dx for monochrome. It is typical in AGP chipsets to shadow this bit and properly steer I/O cycles to the proper bus for operation where a MDA exists on another bus such as ISA. 0 = Select 3Bxh I/O address (MDA emulation) (default). 1 = Select 3Dxh I/O address (CGA emulation).

In standard VGA modes using the analog VGA connector, bits 7 and 6 indicate which of the three standard VGA vertical resolutions the standard VGA display should use. Extended modes, including those with a vertical resolution of 480 scan lines, may use a setting of 0 for both of these bits. Different connector standards and timing standards specify the proper use of sync polarity. This setting was "reserved" in the VGA standard.

Analog CRT Display Sync Polarities

V	H	Display	Horizontal Frequency	Vertical Frequency
P	P	200 Line	15.7 KHz	60 Hz
N	P	350 Line	21.8 KHz	60 Hz
P	N	400 Line	31.5 KHz	70 Hz
N	N	480 Line	31.5 KHz	60 Hz

Sequencer Registers

The sequencer registers are accessed via either I/O space or Memory space. To access registers the VGA Sequencer Index register (SRX) at I/O address 3C4h (or memory address 3C4h) is written with the index of the desired register. Then the desired register is accessed through the data port for the sequencer registers at I/O address 3C5 (or memory address 3C5).

SRX - Sequencer Index

I/O (and Memory Offset) Address: 3C4h

Default: 00h

Attributes: Read/Write

Bit	Description
7:3	**Reserved.** Read as 0s.
2:0	**Sequencer Index.** This field contains a 3-bit Sequencer Index value used to access sequencer data registers at indices 0 through 7.

SR00 - Sequencer Reset

I/O (and Memory Offset) Address: 3C5h(Index=00h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:2	**Reserved.** Read as 0.
1	**Reserved.** Reserved for VGA compatibility (was reset).
0	**Reserved.** Reserved for VGA compatibility. (was reset)

SR01 - Clocking Mode

I/O (and Memory Offset) Address: 3C5h (Index=01h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
5	**Screen Off.** 0 = Normal Operation (default). 1 = Disables video output (blanks the screen) and turns off display data fetches. Synchronization pulses to the display, however, are maintained. Setting this bit to 1 had been used as a way to more rapidly update and improve CPU access performance to the frame buffer during VGA modes. In non-VGA modes (VGA Disable=1), this bit has no effect. Before the VGA is disabled through the MMIO VGA control register, this bit should be set to stop the memory accesses from the display. The following sequence must be used when disabling the VGA plane. 1. Write SR01 to set bit 5 = 1 to disable video output. 2. Wait for 100us. 3. Disable the VGA plane via Bit 31 of the MMIO VGA control register (location found in the MMIO display register programming specification).
4	**Shift 4.** 0 = Load video shift registers every 1 or 2 character clocks (depending on bit 2 of this register) (default). 1 = Load shift registers every 4th character clock.
3	**Dot Clock Divide.** Setting this bit to 1 stretches doubles all horizontal timing periods that are specified in the VGA horizontal CRTC registers. This bit is used in standard VGA 40-column text modes to stretch timings to create horizontal resolutions of either 320 or 360 pixels (as opposed to 640 or 720 pixels, normally used in standard VGA 80-column text modes). The effect of this is that there will actually be twice the number of pixels sent to the display per line. 0 = Pixel clock is left unaltered (used for 640 (720) pixel modes); (default). 1 = Pixel clock divided by 2 (used for 320 (360) pixel modes).
2	**Shift Load.** Bit 4 of this register must be 0 for this bit to be effective. 0 = Load video data shift registers every character clock (default). 1 = Load video data shift registers every other character clock.
1	**Reserved.** Read as 0.

Bit	Descriptions
0	**8/9 Dot Clocks.** This bit determines whether a character clock is 8 or 9 dot clocks long if clock doubling is disabled and 16 or 18 clocks if it is. This also changes the interpretation of the pixel panning values (see chart). An additional control bit determines if this bit is to be ignored and 8-dot characters are to be used always. The 9-dot disable would be used when doubling the horizontal pixels on a 1280 wide display or non-doubling on a 640 wide display. Panning however will occur according to the expected outcome. 0 = 9 dot clocks (9 horizontal pixels) per character in text modes with a horizontal resolution of 720 pixels. 1 = 8 dot clocks (8 horizontal pixels) per character in text or graphics modes with a horizontal resolution of 640 pixels.

SR02 - Plane/Map Mask

I/O (and Memory Offset) Address: 3C5h (Index=02h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:4	**Reserved.** Read as 0s.
3:0	**Memory Planes [3:0] Processor Write Access Enable.** In both the Odd/Even Mode and the Chain 4 Mode, these bits still control access to the corresponding color plane. 0 = Disable. 1 = Enable. This register is referred to in the VGA standard as the Map Mask Register.

SR03 - Character Font

I/O (and Memory Offset) Address: 3C5h (index=03h)

Default: 00h

Attributes: Read/Write

In text modes, bit 3 of the video data's attribute byte normally controls the foreground intensity. This bit may be redefined to control switching between character sets. This latter function is enabled whenever there is a difference in the values of the Character Font Select A and the Character Font Select B bits. If the two values are the same, the character select function is disabled and attribute bit 3 controls the foreground intensity.

Bit 1 of the Memory Mode Register (SR04) must be set to 1 for the character font select function of this register to be active. Otherwise, only character maps 0 and 4 are available.

<table>
<tr><th>Bit</th><th>Descriptions</th></tr>
<tr><td>7:6</td><td>Reserved. Read as 0s.</td></tr>
<tr><td>3:2,5</td><td>Character Map Select Bits for Character Map B. These three bits are used to select the character map (character generator tables) to be used as the secondary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [3:2,5]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
<tr><td>10,0</td><td>2</td><td>5th 8KB of plane 2 at offset 32K</td></tr>
<tr><td>10,1</td><td>6</td><td>6th 8KB of plane 2 at offset 40K</td></tr>
<tr><td>11,0</td><td>3</td><td>7th 8KB of plane 2 at offset 48K</td></tr>
<tr><td>11,1</td><td>7</td><td>8th 8KB of plane 2 at offset 56K</td></tr>
</table></td></tr>
<tr><td>1:0,4</td><td>Character Map Select Bits for Character Map A. These three bits are used to select the character map (character generator tables) to be used as the primary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [1:0,4]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
<tr><td>10,0</td><td>2</td><td>5th 8KB of plane 2 at offset 32K</td></tr>
<tr><td>10,1</td><td>6</td><td>6th 8KB of plane 2 at offset 40K</td></tr>
<tr><td>11,0</td><td>3</td><td>7th 8KB of plane 2 at offset 48K</td></tr>
<tr><td>11,1</td><td>7</td><td>8th 8KB of plane 2 at offset 56K</td></tr>
</table></td></tr>
</table>

SR04 - Memory Mode Register

I/O (and Memory Offset) Address: 3C5h (index=04h)

Default: 00h

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3	**Chain 4 Mode.** The selections made by this bit affect both CPU Read and write accesses to the frame buffer. 0 = The manner in which the frame buffer memory is mapped is determined by the setting of bit 2 of this register (default). 1 = The frame buffer memory is mapped in such a way that the function of address bits 0 and 1 are altered so that they select planes 0 through 3. This setting is used in mode x13 to allow all four planes to be accessed via sequential addresses.
2	**Odd/Even Mode.** Bit 3 of this register must be set to 0 for this bit to be effective. The selections made by this bit affect only non-paged CPU accesses to the frame buffer through the VGA aperture. 0 = The frame buffer memory is mapped in such a way that the function of address bit 0 such that even addresses select planes 0 and 2 and odd addresses select planes 1 and 3 (default). 1 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).
1	**Extended Memory Enable.** This bit must be set to 1 to enable the selection and use of character maps in plane 2 via the Character Map Select Register (SR03). 0 = Disable CPU accesses to more than the first 64KB of VGA standard memory (default). 1 = Enable CPU accesses to the rest of the 256KB total VGA memory beyond the first 64KB.
0	**Reserved.** Read as 0.

SR07 - Horizontal Character Counter Reset

I/O (and Memory Offset) Address: 3C5h (index=07h)

Default: 00h

Attributes: Read/Write

For standard VGAs, writing this register (with any data) causes the horizontal character counter to be held in reset (the character counter output will remain 0). It remained in reset until a write occurred to any other sequencer register location with SRX set to an index of 0 through 6. In this implementation that sequence has no such special effect.

The vertical line counter is clocked by a signal derived from the horizontal display enable (which does not occur if the horizontal counter is held in reset). Therefore, if a write occurs to this register during the vertical retrace interval, both the horizontal and vertical counters will be set to 0. A write to any other sequencer register location (with SRX set to an index of 0 through 6) may then be used to start both counters with reasonable synchronization to an external event via software control. Although this was a standard VGA register, it was not documented.

Bit	Description
7:0	**Horizontal Character Counter.**

Graphics Controller Registers

The graphics controller registers are accessed via either I/O space or Memory space. Accesses to the registers of the VGA Graphics Controller are done through the use of address 3CEh (or memory address 3CEh) written with the index of the desired register. Then the desired register is accessed through the data port for the graphics controller registers at I/O address 3CFh (or memory address 3CFh). Indexes 10 and 11 should only be accessed through the I/O space only.

GRX - GRX Graphics Controller Index Register

I/O (and Memory Offset) Address: 3CEh

Default: 000UUUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:0	**Graphics Controller Register Index.** This field selects any one of the graphics controller registers (GR00-GR18) to be accessed via the data port at I/O (or memory offset) location 3CFh.

GR00 - Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (index=00h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Set/Reset Plane [3:0].** When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 0, all 8 bits of each byte of each memory plane are set to either 1 or 0 as specified in the corresponding bit in this register, if the corresponding bit in the Enable Set/Reset Register (GR01) is set to 1. When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 3, all CPU data written to the frame buffer is rotated, then logically ANDed with the contents of the Bit Mask Register (GR08), and then treated as the addressed data's bit mask, while value of these four bits of this register are treated as the color value.

GR01 - Enable Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (Index=01h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Enable Set/Reset Plane [3:0].** This register works in conjunction with the Set/Reset Register (GR00). The Write Mode bits (bits 0 and 1) must be set for Write Mode 0 for this register to have any effect. 0 = The corresponding memory plane can be read from or written to by the CPU without any special bitwise operations taking place. 1 = The corresponding memory plane is set to 0 or 1 as specified in the Set/Reset Register (GR00).

GR02 - Color Compare Register

I/O (and Memory Offset) Address: 3CFh (Index=02h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Color Compare Plane [3:0].** When the Read Mode bit (bit 3) of the Graphics Mode Register (GR05) is set to select Read Mode 1, all 8 bits of each byte of each of the 4 memory planes of the frame buffer corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison, wherein value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.

GR03 - Data Rotate Register

I/O (and Memory Offset) Address: 3CFh (Index=03h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:3	**Function Select.** These bits specify the logical function (if any) to be performed on data that is meant to be written to the frame buffer (using the contents of the memory read latch) just before it is actually stored in the frame buffer at the intended address location. 00 = Data being written to the frame buffer remains unchanged, and is simply stored in the frame buffer. 01 = Data being written to the frame buffer is logically ANDed with the data in the memory read latch before it is actually stored in the frame buffer. 10 = Data being written to the frame buffer is logically ORed with the data in the memory read latch before it is actually stored in the frame buffer. 11 = Data being written to the frame buffer is logically XORed with the data in the memory read latch before it is actually stored in the frame buffer.
2:0	**Rotate Count.** These bits specify the number of bits to the right to rotate any data that is meant to be written to the frame buffer just before it is actually stored in the frame buffer at the intended address location.

GR04 - Read Plane Select Register

I/O (and Memory Offset) Address: 3CFh (Index=04h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**Read Plane Select.** These two bits select the memory plane from which the CPU reads data in Read Mode 0. In Odd/Even Mode, bit 0 of this register is ignored. In Chain 4 Mode, both bits 1 and 0 of this register are ignored. The four memory planes are selected as follows: 00 = Plane 0

Bit	Description
	01 = Plane 1 10 = Plane 2 11 = Plane 3 These two bits also select which of the four memory read latches may be read via the Memory read Latch Data Register (CR22). The choice of memory read latch corresponds to the choice of plane specified in the table above. The Memory Read Latch Data register and this additional function served by 2 bits are features of the VGA standard that were never documented.

GR05 - Graphics Mode Register

I/O (and Memory Offset) Address: 3CFh (Index=05h)

Default: 0UUU U0UUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Shift Register Control.** In standard VGA modes, pixel data is transferred from the 4 graphics memory planes to the palette via a set of 4 serial output bits. These 2 bits of this register control the format in which data in the 4 memory planes is serialized for these transfers to the palette. **Bits [6:5]=00** One bit of data at a time from parallel bytes in each of the 4 memory planes is transferred to the palette via the 4 serial output bits, with 1 of each of the serial output bits corresponding to a memory plane. This provides a 4-bit value on each transfer for 1 pixel, making possible a choice of 1 of 16 colors per pixel. Serial Out / 1st Xfer / 2nd Xfer / 3rd Xfer / 4th Xfer / 5th Xfer / 6th Xfer / 7th Xfer / 8th Xfer Bit 3 / plane3 bit7 / plane3 bit6 / plane3 bit5 / plane3 bit4 / plane3 bit3 / plane3 bit2 / plane3 bit1 / plane3 bit0 Bit 2 / plane2 bit7 / plane2 bit6 / plane2 bit5 / plane2 bit4 / plane2 bit3 / plane2 bit2 / plane2 bit1 / plane2 bit0 Bit 1 / plane1 bit7 / plane1 bit6 / plane1 bit5 / plane1 bit4 / plane1 bit3 / plane1 bit2 / plane1 bit1 / plane1 bit0 Bit 0 / plane0 bit7 / plane0 bit6 / plane0 bit5 / plane0 bit4 / plane0 bit3 / plane0 bit2 / plane0 bit1 / plane0 bit0

Bit	Description
	Bits [6:5]=01 Two bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that alternates per byte between memory planes 0 and 2, and memory planes 1 and 3. First the even-numbered and odd-numbered bits of a byte in memory plane 0 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of a byte in memory plane 2 are transferred via serial output bits 2 and 3. Next, the even-numbered and odd-numbered bits of a byte in memory plane 1 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of memory plane 3 are transferred via serial out bits 1 and 3. This provides a pair of 2-bit values (one 2-bit value for each of 2 pixels) on each transfer, making possible a choice of 1 of 4 colors per pixel. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane2 bit7 plane2 bit5 plane2 bit3 plane2 bit1 plane3 bi7t plane3 bit5 plane3 bit3 plane3 bit1 Bit 2 plane2 bit6 plane2 bit4 plane2 bit2 plane2 bit0 plane3 bit6 plane3 bit4 plane3 bit2 plane3 bit0 Bit 1 plane0 bit7 plane0 bit5 plane0 bit3 plane0 bit1 plane1 bit7 plane1 bit5 plane1 bit3 plane1 bit1 Bit 0 plane0 bit6 plane0 bit4 plane0 bit2 plane0 bit0 plane1 bit6 plane1 bit4 plane1 bit2 plane1 bit0 This alternating pattern is meant to accommodate the use of the Odd/Even mode of organizing the 4 memory planes, which is used by standard VGA modes 2h and 3h. **Bits [6:5]=1x** Four bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that iterates per byte through memory planes 0 through 3. First the 4 most significant bits of a byte in memory plane 0 are transferred via the 4 serial output bits, followed by the 4 least significant bits of the same byte. Next, the same transfers occur from the parallel byte in memory planes 1, 2 and lastly, 3. Each transfer provides either the upper or lower half of an 8 bit value for the color for each pixel, making possible a choice of 1 of 256 colors per pixel. This is the setting used in mode x13. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane0 plane0 plane1 plane1 plane2 plane2 plane3 plane3

Bit	Description
	<table><tr><td></td><td>bit7</td><td>bit3</td><td>bit7</td><td>bit3</td><td>bit7</td><td>bit3</td><td>bit7</td><td>b3it</td></tr><tr><td>Bit 2</td><td>plane0 bit6</td><td>plane0 bit2</td><td>plane1 bit6</td><td>plane1 bit2</td><td>plane2 bit6</td><td>plane2 bit2</td><td>plane3 bit6</td><td>plane3 bit2</td></tr><tr><td>Bit 1</td><td>plane0 bit5</td><td>plane0 bit1</td><td>plane1 bit5</td><td>plane1 bit1</td><td>plane2 bit5</td><td>plane2 bit1</td><td>plane3 bit5</td><td>plane3 bit1</td></tr><tr><td>Bit 0</td><td>plane0 bit4</td><td>plane0 bit0</td><td>plane1 bit4</td><td>plane1 bit0</td><td>plane2 bit4</td><td>plane2 bit0</td><td>plane3 bit4</td><td>plane3 bit0</td></tr></table> This pattern is meant to accommodate mode 13h, a standard VGA 256-color graphics mode.
4	**Odd/Even Mode.** 0 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02). 1 = The frame buffer is mapped in such a way that the function of address bit 0 is such that even addresses select memory planes 0 and 2 and odd addresses select memory planes 1 and 3. This works in a way that is the inverse of (and is normally set to be the opposite of) bit 2 of the Memory Mode Register (SR02).
3	**Read Mode.** 0 = During a CPU read from the frame buffer, the value returned to the CPU is the memory plane selected by bits 1 and 0 of the Read Plane Select Regist 1 = During a CPU read from the frame buffer, all 8 bits of the byte in each of the 4 memory planes corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison. A value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all 4 of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.
2	**Reserved.** Read as 0.

Bit	Description
1:0	**Write Mode.** 00 = Write Mode 0 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written with the CPU write data after it has been rotated by the number of counts specified in the Data Rotate Register (GR03). If, however, the bit(s) in the Enable Set/Reset Register (GR01) corresponding to one or more of the memory planes is set to 1, then those memory planes will be written to with the data stored in the corresponding bits in the Set/Reset Register (GR00). 01 = Write Mode 1 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written to with the data stored in the memory read latches. (The memory read latches stores an unaltered copy of the data last read from any location in the frame buffer.) 10 = Write Mode 2 - During a CPU write to the frame buffer, the least significant 4 data bits of the CPU write data is treated as the color value for the pixels in the addressed byte in all 4 memory planes. The 8 bits of the Bit Mask Register (GR08) are used to selectively enable or disable the ability to write to the corresponding bit in each of the 4 memory planes that correspond to a given pixel. A setting of 0 in a bit in the Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with value of their counterparts in the memory read latches. A setting of 1 in a Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with the 4 bits taken from the CPU write data to thereby cause the pixel corresponding to these bits to be set to the color value. 11 = Write Mode 3 - During a CPU write to the frame buffer, the CPU write data is logically ANDed with the contents of the Bit Mask Register (GR08). The result of this ANDing is treated as the bit mask used in writing the contents of the Set/Reset Register (GR00) are written to addressed byte in all 4 memory planes.

GR06 - Miscellaneous Register

I/O (and Memory Offset) Address: 3CFh (Index=06h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0s.
3:2	**Memory Map Mode.** These 2 bits control the mapping of the VGA address range for frame buffer into the CPU address space as follows: 00 = A0000h - BFFFFh 01 = A0000h - AFFFFh 10 = B0000h - B7FFFh 11 = B8000h - BFFFFh This function is used in standard VGA modes, extended VGA modes (132 column text), and in non-VGA modes (hi-res). 132 column text modes are no longer supported. VGA aperture memory accesses are also controlled by the PCI configuration Memory Enable bit and MSR<1>. For accesses using GR10 and GR11 to paged VGA RAM or to device MMIO registers, set these bits to 01 to select the (A0000-AFFFF) range. The CPU must map this memory as uncacheable (UC).
1	**Chain Odd/Even.** This bit provides the ability to alter the interpretation of address bit A0, so that it may be used in selecting between the odd-numbered memory planes (planes 1 and 3) and the even-numbered memory planes (planes 0 and 2). 0 = A0 functions normally. 1 = A0 is switched with a high order address bit, in terms of how it is used in address decoding. The result is that A0 is used to determine which memory plane is being accessed (A0=0 for planes 0 and 2 and A0=1 for planes 1 and 3).

Bit	Description
0	**Graphics/Text Mode.** This is one of two bits that are used to determine if the VGA is operating in text or graphics modes. The other bit is in AR10[0], these two bits need to be programmed in a consistent manner to achieve the proper results. 0 = Text mode. 1 = Graphics mode.

GR07 - Color Don't Care Register

I/O (and Memory Offset) Address: 3CFh (Index=07h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Ignore Color Plane [3:0].** These bits have effect only when bit 3 of the Graphics Mode Register (GR05) is set to 1 to select read mode 1. 0 = The corresponding bit in the Color Compare Register (GR02) will not be included in color comparisons. 1 = The corresponding bit in the Color Compare Register (GR02) is used in color comparisons.

GR08 - Bit Mask Register

I/O (and Memory Offset) Address: 3CFh (Index=08h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Bit Mask.** 0 = The corresponding bit in each of the 4 memory planes is written to with the corresponding bit in the memory read latches. 1 = Manipulation of the corresponding bit in each of the 4 memory planes via other mechanisms is enabled. This bit mask applies to any writes to the addressed byte of any or all of the 4 memory planes, simultaneously. This bit mask is applicable to any data written into the frame buffer by the CPU, including data that is also subject to rotation, logical functions (AND, OR, XOR), and Set/Reset. To perform a proper read-modify-write cycle into frame buffer, each byte must first be read from the frame buffer by the CPU (and this will cause it to be stored in the memory read latches), this Bit Mask Register must be set, and the new data then written into the frame buffer by the CPU.

GR10 - Address Mapping

I/O (avoid MMIO access) Address: 3CFh (Index=10h)

Default: 00h

Attributes: R/W

This register should only be accessed using I/O operations and never be accessed through the A/B segment addressing map, I/O space register map, or direct MMIO operations.

Bit	Description
7:4	**Page Select Extension - Unused** These bits form the upper bits of a 12-bit page selection value. When combined with the GR11 <7:0> bits they define the offset into stolen memory to the 64KB page that is accessible via the VGA Memory paging mechanism. These bits are ignored.
3	**Reserved**

Bit	Description
2:1	**Paging Map Target.** When paging is enabled, these bits determine the target for data cycle accesses through the VGA memory aperture. VGA graphics memory starts from the base of graphics data stolen memory defined in the PCI configuration BDSM register. VGA display uses the first four 64KB pages of VGA graphics memory. 00 = VGA Graphics Memory 01 = Reserved 10 = Reserved 11 = Reserved
0	**Page Mapping Enable.** This mode allows the mapping of the VGA memory address space. Once this is enabled, no VGA memory address swizzle will be performed, addresses are directly mapped to memory. A single paging register is used to map the 64KB [A0000:AFFFF] window. An internal address is generated using GR11 as the address lines extension to the lower address lines of the access A[15:2]. When mapping is enabled, the B0000:BFFFF area must be disabled using GR06<3:2>=01. The use of addresses in the A0000-BFFFF range require that both the graphics device PCI configuration memory enable and MSR<1> be enabled. 0 = Disable (default) 1 = Enable

GR11 - Page Selector

I/O (avoid MMIO access) Address: 3CFh (Index=11h)

Default: 00h

Attributes: R/W

Bit	Description
7	**Reserved**
6:0	**Page Select.** When concatenated with the GR10<7:4> bits, selects a 64KB window within target area when Page Mapping is enabled (GR10[0]=1). This requires that the graphics device PCI configuration space memory enable, the GR06<3:2> bits to be 01 (select A0000-AFFFF only), and the MSR<1:1> bit to be set. This register provides the Address[22:16] bits for the access. VGA paging of frame buffer memory is for non-VGA packed modes only and should not be enabled when using basic VGA modes. This register should only be accessed using I/O operations.

GR18 - Software Flags

I/O (and Memory Offset) Address: 3CFh (Index=18h)

Default: 00

Attributes: R/W

Bit	Description
7:0	**Software Flags.** Used as scratch pad space in video BIOS. These bits are separate from the bits which appear in the memory mapped IO space. They are used specifically by the SMI BIOS which does not have access to memory mapped IO at the time they are required. These register bits have no effect on H/W operation.

Attribute Controller Registers

Unlike the other sets of indexed registers, the attribute controller registers are not accessed through a scheme employing entirely separate index and data ports. I/O address 3C0h (or memory address 3C0h) is used both as the read and write for the index register, and as the write address for the data port. I/O address 3C1h (or memory address 3C1h) is the read address for the data port.

To write to the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is written to the very same I/O (memory) address. A flip-flop alternates with each write to I/O address 3C0h (or memory address 3C0h) to change its function from writing the index to writing the actual data, and back again. This flip-flop may be deliberately set so that I/O address 3C0h (or memory address 3C0h) is set to write to the index (which provides a way to set it to a known state) by performing a read operation from Input Status Register 1 (ST01) at I/O address 3BAh (or memory address 3BAh) or 3DAh (or memory address 3DAh), depending on whether the graphics system has been set to emulate an MDA or a CGA as per MSR[0].

To read from the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is read from I/O address 3C1h (or memory address 3C1h). A read operation from I/O address 3C1h (or memory address 3C1h) does not reset the flip-flop to writing to the index. Only a write to 3C0h (or memory address 3C0h) or a read from 3BAh or 3DAh (or memory address 3BAh or 3DAh), as described above, will toggle the flip-flop back to writing to the index.

ARX - Attribute Controller Index Register

I/O (and Memory Offset) Address: 3C0h

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0s.
5	**Video Enable.** In the VGA standard, this is called the "Palette Address Source" bit. Clearing this bit will cause the VGA display data to become all 00 index values. For the default palette, this will cause a black screen. The video timing signals continue. Another control bit will turn video off and stop the data fetches. 0 = Disable. Attribute controller color registers (AR[00:0F]) can be accessed by the CPU. 1 = Enable. Attribute controller color registers (AR[00:0F]) are inaccessible by the CPU.
4:0	**Attribute Controller Register Index.** These five bits are used to select any one of the attribute controller registers (AR[00:14]), to be accessed.

AR[00:0F] - Palette Registers [0:F]

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=00h-0Fh)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:0	**Palette Bits P[5:0].** In each of these 16 registers, these are the lower 6 of 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors available to be selected in the palette. Bits 3 and 2 of the Color Select Register (AR14) supply bits P7 and P6 for the values contained in all 16 of these registers. Bits 1 and 0 of the Color Select Register (AR14) can also replace bits P5 and P4 for the values contained in all 16 of these registers, if bit 7 of the Mode Control Register (AR10) is set to 1.

AR10 - Mode Control Register

I/O (and Memory Offset) Address:	Read at 3C1h and Write at 3C0h; (index=10h)
Default:	UUh (U=Undefined)
Attributes:	Read/Write

Bit	Description
7	**Palette Bits P5, P4 Select.** **0** = P5 and P4 for each of the 16 selected colors (for modes that use 16 colors) are individually provided by bits 5 and 4 of their corresponding Palette Registers (AR[00:0F]). **1** = P5 and P4 for all 16 of the selected colors (for modes that use 16 colors) are provided by bits 1 and 0 of Color Select Register (AR14).
6	**Pixel Width/Clock Select.** **0** = Six bits of video data (translated from 4 bits via the palette) are output every dot clock. **1** = Two sets of 4 bits of data are assembled to generate 8 bits of video data which is output every other dot clock, and the Palette Registers (AR[00:0F]) are bypassed. This bit is set to 0 for all of the standard VGA modes, except mode 13h.
5	**Pixel Panning Compatibility.** **0** = Scroll both the upper and lower screen regions horizontally as specified in the Pixel Panning Register (AR13). **1** = Scroll only the upper screen region horizontally as specified in the Pixel Panning Register (AR13). This bit has application only when split-screen mode is being used, where the display area is divided into distinct upper and lower regions which function somewhat like separate displays.
4	**Reserved.** Read as 0.
3	**Enable Blinking/Select Background Intensity.** **0** = Disables blinking in graphics modes, and for text modes, sets bit 7 of the character attribute bytes to control background intensity, instead of blinking. **1** = Enables blinking in graphics modes and for text modes, sets bit 7 of the character attribute bytes to control blinking, instead of background intensity. The blinking rate is derived by dividing the VSYNC signal. The Blink Rate Control field of the VGA control register defines the blinking rate.

Bit	Description
2	**Enable Line Graphics Character Code.** **0** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the background of the character of which the given pixel is a part. **1** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the 8th pixel if the character of which the given pixel is a part. This setting is intended to accommodate the line-drawing characters of the PC's extended ASCII character set -- characters with an extended ASCII code in the range of B0h to DFh. In some literature describing the VGA standard, the range of extended ASCII codes that are said to include the line-drawing characters is mistakenly specified as C0h to DFh, rather than the correct range of B0h to DFh.
1	**Select Display Type.** **0** = Attribute bytes in text modes are interpreted as they would be for a color display. **1** = Attribute bytes in text modes are interpreted as they would be for a monochrome display.
0	**Graphics/Alphanumeric Mode.** This bit (along with GR06[0]) select either graphics mode or text mode. These two bits must be programmed in a consistent manner to achieve the desired results. **0** = Alphanumeric (text) mode. **1** = Graphics mode.

AR11 - Overscan Color Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=11h)

Default: UUh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:0	**Overscan.** These 8 bits select the overscan (border) color index value. The actual border color will be determined by the contents of the palette at the selected index. The border color is displayed between the end of active and the beginning of blank or the end of blank and the beginning of active on CRT type devices driven from the DAC output port. For native VGA modes on digital display ports there is the option of including the border in the active region or not depending on a control bit in the port control register. For centered VGA modes, the VGA control register determines if the border is included in the centered region or not. For monochrome displays, this value should be set to 00h.

AR12 - Memory Plane Enable Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=12h)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:4	**Video Status Mux.** These 2 bits are used to select 2 of the 8 possible palette bits (P7-P0) to be made available to be read via bits 5 and 4 of the Input Status Register 1 (ST01). The table below shows the possible choices. Bit [5:4] / ST01 Bit 5 / ST01 Bit 4 00 / P2 (default) / P0 (default) 01 / P5 / P4 10 / P3 / P1 11 / P7 / P6 These bits are typically unused by current software; they are provided for EGA compatibility.
3:0	**Enable Plane [3:0].** These 4 bits individually enable the use of each of the 4 memory planes in providing 1 of the 4 bits used in video output to select 1 of 16 possible colors from the palette to be displayed. 0 = Disable the use of the corresponding memory plane in video output to select colors, forcing the bit that the corresponding memory plane would have provided to a value of 0. 1 = Enable the use of the corresponding memory plane in video output to select colors. AR12 is referred to in the VGA standard as the Color Plane Enable Register.

AR13 - Horizontal Pixel Panning Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=13h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:0	**Horizontal Pixel Shift 3-0.** This field holds a 4-bit value that selects the number of pixels by which the image is shifted horizontally to the left. This function is available in both text and graphics modes and allows for pixel panning. In text modes with a 9-pixel wide character box, the image can be shifted up to 9 pixels to the left. In text modes with an 8-pixel wide character box, and in graphics modes other than those with 256 colors, the image can be shifted up to 8 pixels to the left. A pseudo 9-bit mode is when the 9-dot character is selected but overridden by the VGA control bit. In standard VGA mode 13h (where bit 6 of the Mode Control Register, AR10, is set to 1 to support 256 colors), bit 0 of this register must remain set to 0, and the image may be shifted up to only 4 pixels to the left. In this mode, the number of pixels by which the image is shifted can be further controlled using bits 6 and 5 of the Preset Row Scan Register (CR08).

Number of Pixels Shifted				
Bits [3:0]	**9-dot**	**Pseudo 9-dot**	**8-dot**	**256-Color**
0	1	1	0	0
1	2	2	1	Undefined
2	3	3	2	1
3	4	4	3	Undefined
4	5	5	4	2
5	6	6	5	Undefined
6	7	7	6	3
7	8	7	7	Undefined
8	0	0	Undefined	Undefined

AR14 - Color Select Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=14h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:2	**Palette Bits P[7:6].** These are the 2 upper-most of the 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors contained in the palette. These 2 bits are common to all 16 sets of bits P5 through P0 that are individually supplied by Palette Registers 0-F (AR[00:0F]).
1:0	**Alternate Palette Bits P[5:4].** These 2 bits can be used as an alternate version of palette bits P5 and P4. Unlike the P5 and P4 bits that are individually supplied by Palette Registers 0-F (AR[00:0F]), these 2 alternate palette bits are common to all 16 of Palette Registers. Bit 7 of the Mode Control Register (AR10) is used to select between the use of either the P5 and P4 bits that are individually supplied by the 16 Palette Registers or these 2 alternate palette bits.

VGA Color Palette Registers

In devices that have multiple display pipes, there is one palette for each display pipe. These palettes are the same for VGA modes and non-VGA modes. Accesses through VGA register methods can optionally read or write from either one.

For each palette, the color data stored in these 256 color data positions can be accessed only through a complex sub-addressing scheme, using a data register and two index registers. The Palette Data Register at I/O address 3C9h (or memory address offset 3C1h) is the data port. The Palette Read Index Register at I/O address 3C7h (or memory address offset 3C7h) and the Palette Write Index Register at I/O address 3C8h (or memory address offset 3C8h) are the two index registers. The Palette Read Index Register is the index register that is used to choose the color data position that is to be read from via the data port, while the Palette Write Index Register is the index register that is used to choose the color data position that is to be written to through the same data port. This arrangement allows the same data port to be used for reading from and writing to two different color data positions. Reading and writing the color data at a color data position involves three successive reads or writes since the color data stored at each color data position consists of three bytes.

To read a palette color data position, the index of the desired color data position must first be written to the Palette Read Index Register. Then all three bytes of data in a given color data position may be read at the Palette Data Register. The first byte read from the Palette Data Register retrieves the 8-bit value specifying the intensity of the red color component. The second and third bytes read are the corresponding 8-bit values for the green and blue color components respectively. After completing the third read operation, the Palette Read Index Register is automatically incremented so that the data of the next color data position becomes accessible for being read. This allows the contents of all of the 256 color data positions of the palette to be read in sequence. This is done by specifying only the index of the 0th color data position in the Palette Read Index Register, and then simply performing 768 successive reads from the Palette Data Register.

Writing a color data position, entails a very similar procedure. The index of the desired color data position must first be written to the Palette Write Index Register. Then all three bytes of data to specify a given color may be written to the Palette Data Register. The first byte written to the Palette Data Register specifies the intensity of the red color component, the second byte specifies the intensity for the green color component, and the third byte specifies the same for the blue color component. One important detail is that all three of these bytes must be written before the hardware will actually update these three values in the given color data position. When all three bytes have been written, the Palette Write Index Register is automatically incremented so that the data of the next color data position becomes accessible for being written. This allows the contents of all of the 256 color data positions of the palette to be written in sequence. This is done by specifying only the index of the 0th color data position in the Palette Write Index Register, and then simply performing 768 successive writes to the Palette Data Register.

DACMASK - Pixel Data Mask Register

I/O (and Memory Offset) Address: 3C6h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Pixel Data Mask.** In indexed-color mode, the 8 bits of this register are logically ANDed with the 8 bits of pixel data received from the frame buffer for each pixel. The result of this ANDing process becomes the actual index used to select color data positions within the palette. This has the effect of limiting the choice of color data positions that may be specified by the incoming 8-bit data. 0 = Corresponding bit in the resulting 8-bit index being forced to 0. 1 = Allows the corresponding bit in the resulting index to reflect the actual value of the corresponding bit in the incoming 8-bit pixel data.

DACSTATE - DAC State Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Read Only

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**DACState.** This field indicates which of the two index registers was most recently written. **Bits [1:0]** Index Register Indicated 00 = Palette Write Index Register at I/O Address 3C7h (default) 01 = Reserved 10 = Reserved 11 = Palette Read Index Register at I/O Address 3C8h

DACRX - Palette Read Index Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Read Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being read from via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been read. A write to this register will abort a uncompleted palette write sequence. This register allows access to the palette even when running non-VGA display modes.

DACWX - Palette Write Index Register

I/O (and Memory Offset) Address: 3C8h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Write Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being written via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been written. This register allows access to the palette even when running non-VGA display modes.

DACDATA - Palette Data Register

I/O (and Memory Offset) Address: 3C9h

Default: Undefined

Attributes: Read/Write

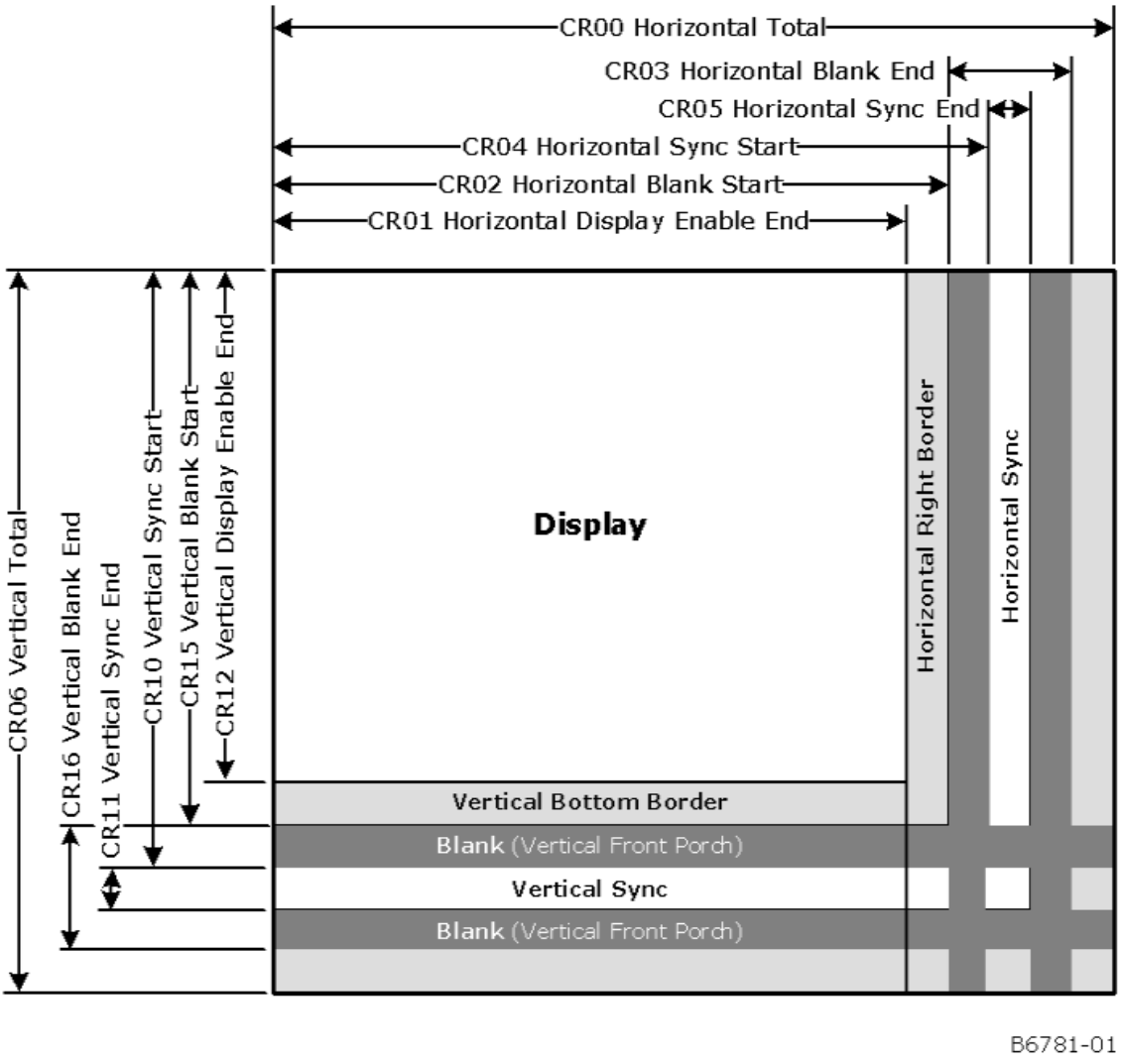
Bit	Description
7:0	**Palette Data.** This byte-wide data port provides read or write access to the three bytes of data of each color data position selected using the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX). The three bytes in each color data position are read or written in three successive read or write operations. The first byte read or written specifies the intensity of the red component of the color specified in the selected color data position. The second byte is for the green component, and the third byte is for the blue component. When writing data to a color data position, all three bytes must be written before the hardware will actually update the three bytes of the selected color data position. When reading or writing to a color data position, ensure that neither the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX) are written to before all three bytes are read or written. A write to either of these two registers causes the circuitry that automatically cycles through providing access to the bytes for red, green and blue components to be reset such that the byte for the red component is the one that will be accessed by the next read or write operation via this register. This register allows access to the palette even when running non-VGA display modes. Writes to the palette can cause sparkle if not done during inactive video periods. This sparkle is caused by an attempt to write and read the same address on the same cycle. Anti-sparkle circuits will substitute the previous pixel value for the read output.

CRT Controller Register

For native VGA modes, the CRTC registers determine the display timing that is to be used. In centered VGA modes, these registers determine the size of the VGA image that is to be centered in the larger timing generator defined rectangle.

The CRT controller registers are accessed by writing the index of the desired register into the CRT Controller Index Register at I/O address 3B4h or 3D4h, depending on whether the graphics system is configured for MDA or CGA emulation. The desired register is then accessed through the data port for the CRT controller registers located at I/O address 3B5h or 3D5h, again depending upon the choice of MDA or CGA emulation as per MSR[0]. For memory mapped accesses, the Index register is at 3B4h (MDA mode) or 3D3h (CGA mode) and the data port is accessed at 3B5h (MDA mode) or 3D5h (CGA mode).

The following figure shows display fields and dimensions and the particular CRxx register that provides the control.



Group 0 Protection: In the original VGA, CR[0:7] could be made write-protected by CR11[7]. In BIOS code, this write protection is set following each mode change. Other protection groups have no current use, and would not be used going forward by the BIOS or by drivers. They are the result of an industry fad some years ago to attempt to write protect other groups of registers; however, all such schemes were chip specific. Only the write protection (Group 0 Protection) is supported.

CRX - CRT Controller Index Register

I/O (and Memory Offset) Address: 3B4h/3D4h

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:0	**CRT Controller Index.** These 7 bits are used to select any one of the CRT controller registers to be accessed via the data port at I/O location 3B5h or 3D5h, depending upon whether the graphics system is configured for MDA or CGA emulation. The data port memory address offsets are 3B5h/3D5h.

CR00 - Horizontal Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=00h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Total.** This register is used to specify the total length of each scan line. This encompasses both the part of the scan line that is within the active display area and the part that is outside of it. Programming this register to a zero has the effect of stopping the fetching of display data. This field should be programmed with a value equal to the total number of character clocks within the entire length of a scan line, minus 5.

CR01 - Horizontal Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=01h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Display Enable End.** This register is used to specify the end of the part of the scan line that is within the active display area relative to its beginning. In other words, this is the horizontal width of the active display area. This field should be programmed with a value equal to the number of character clocks that occur within the horizontal active display area, minus 1. Horizontal display enable will go active at the beginning of each line during vertical active area, it will go inactive based on the programming of this register or the programming of the horizontal total (CR00) register. When this register value is programmed to a number that is larger than the total number of characters on a line, display enable will be active for all but the last character of the horizontal display line.

CR02 - Horizontal Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=02h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Blanking Start.** This register is used to specify the beginning of the horizontal blanking period relative to the beginning of the active display area of a scan line. Horizontal blanking should always be set to start no sooner than after the end of horizontal active. This field should be programmed with a value equal to the number of character clocks that occur on a scan line from the beginning of the active display area to the beginning of the horizontal blanking.

CR03 - Horizontal Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=03h)

Default: 1UUU UUUUb (U=Undefined)

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Reserved.** Values written to this bit are ignored, and to maintain consistency with the VGA standard, a value of 1 is returned when this bit is read. At one time, this bit was used to enable access to certain light pen registers. At that time, setting this bit to 0 provided this access, but setting this bit to 1 was necessary for normal operation.
6:5	**Display Enable Skew Control.** Defines the degree to which the start and end of the active display area are delayed along the length of a scan line to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = no delay 01 = delayed by 1 character clock 10 = delayed by 2 character clocks 11 = delayed by 3 character clocks
4:0	**Horizontal Blanking End Bits [4:0].** This field provides the 5 least significant bits of a 6-bit value that specifies the end of the blanking period relative to its beginning on a single scan line. Bit 7 of the Horizontal Sync End Register (CR05) supplies the most significant bit. This 6-bit value should be programmed to be equal to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02). End of blanking should occur before horizontal total.

CR04 - Horizontal Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=04h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Sync Start** This register is used to specify the position of the beginning of the horizontal sync pulse relative to the start of the active display area on a scan line. This field should be set equal to the number of character clocks that occur from beginning of the active display area to the beginning of the horizontal sync pulse on a single scan line. Horizontal sync should always occur at least 2 clocks after the start of horizontal blank and 2 clocks before the end of horizontal blank. The actual start of sync will also be affected by both the horizontal sync skew register field and whether it is a text or graphics mode.

CR05 - Horizontal Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=05h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Horizontal Blanking End Bit 5.** This bit provides the most significant bit of a 6-bit value that specifies the end of the horizontal blanking period relative to its beginning. Bits [4:0] of Horizontal Blanking End Register (CR03) supplies the 5 least significant bits. See CR03[4:0] for further details. This 6-bit value should be set to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02).
6:5	**Horizontal Sync Delay.** This field defines the degree to which the start and end of the horizontal sync pulse are delayed to compensate for internal pipeline delays. This capability is supplied to implement VGA compatibility. These field describes the delay in terms of a number character clocks. **Bit [6:5] — Amount of Delay** 00 — no delay 01 — delayed by 1 character clock 10 — delayed by 2 character clocks 11 — delayed by 3 character clocks
4:0	**Horizontal Sync End.** This field provides the 5 least significant bits of a 5-bit value that specifies the end of the horizontal sync pulse relative to its beginning. A value equal to the 5 least significant bits of the horizontal character counter value at which time the horizontal retrace signal becomes inactive (logical 0). Thus, this 5-bit value specifies the width of the horizontal sync pulse. To obtain a retrace signal of W, the following algorithm is used: Value of Horizontal Sync start Register (CR04) + width of horizontal retrace signal in character clock units = 5 bit result to be programmed in this field

CR06 - Vertical Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=06h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Vertical Total Bits [7:0].** This field provides the 8 least significant bits of either a 10-bit or 12-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. In standard VGA modes, the vertical total is specified with a 10-bit value. The 8 least significant bits of this value are supplied by these 8 bits of this register, and the 2 most significant bits are supplied by bits 5 and 0 of the Overflow Register (CR07).

CR07 - Overflow Register (Vertical)

I/O (and Memory Offset) Address: 3B5h/3D5h (index=07h)

Default: UU0U UUU0b (U=Undefined)

Attributes: Read/Write (Group 0 Protection on bits [7:5, 3:0])

Bit	Description
7	**Vertical Sync Start Bit 9.** The vertical sync start is a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by this bit and bit 2, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
6	**Vertical Display Enable End Bit 9.** The vertical display enable end is a 10-bit that specifies the number of the last scan line within the active display area. In standard VGA modes, the vertical display enable end is specified with a 10-bit value. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the most and second-most significant bits are supplied by this bit and bit 1, respectively, of this register. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
5	**Vertical Total Bit 9.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by this bit and bit 0, respectively, of this register. This 10-bit value should be programmed equal to the total number of scan lines, minus 2.

Bit	Description
4	**Line Compare Bit 8.** This bit provides the second most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display what data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display what data exists in the frame buffer starting at the first byte of the frame buffer.
3	**Vertical Blanking Start Bit 8.** The vertical blanking start is a 10-bit that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by bit 5 of the Maximum Scan Line Register (CR09) and this bit of this register, respectively. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
2	**Vertical Sync Start Bit 8.** The vertical sync start is a 10-bit value that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by bit 7 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

Bit	Description
1	**Vertical Display Enable End Bit 8.** The vertical display enable end is a 10-bit value that specifies the number of the last scan line within the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the two most significant bits are supplied by bit 6 and this bit, respectively, of this register. This 10-bit or value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
0	**Vertical Total Bit 8.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by bit 5 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the total number of scan lines, minus 2.

CR08 - Preset Row Scan Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=08h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0s.
6:5	**Byte Panning.** This field holds a 2-bit value that selects number of bytes (up to 3) by which the image is shifted horizontally to the left on the screen. This function is available in both text and graphics modes. In text modes with a 9-pixel wide character box, the image can be shifted up to 27 pixels to the left, in increments of 9 pixels. In text modes with an 8-pixel wide character box, and in all standard VGA graphics modes, the image can be shifted up to 24 pixels to the left, in increments of 8 pixels. When the Nine dot disable bit of the VGA control register is set, the pixel shift will be equivalent to the 8-dot mode. The image can be shifted still further, in increments of individual pixels, through the use of bits [3:0] of the Horizontal Pixel Panning Register (AR13).
4:0	**Starting Row Scan Count.** This field specifies which horizontal line of pixels within the character boxes of the characters used on the top-most row of text on the display will be used as the top-most scan line. The horizontal lines of pixels of a character box are numbered from top to bottom, with the top-most line of pixels being number 0. If a horizontal line of the these character boxes other than the top-most line is specified, then the horizontal lines of the character box above the specified line of the character box will not be displayed as part of the top-most row of text characters on the display. Normally, the value specified by these 5 bits should be 0, so that all of the horizontal lines of pixels within these character boxes will be displayed in the top-most row of text, ensuring that the characters in the top-most row of text do not look as though they have been cut off at the top.

Number of Pixels Shifted

Bit [6:5]	9-Pixel Text	8-Pixel Text & Graphics
00	0	0
01	9	8
10	18	16
11	27	24

CR09 - Maximum Scan Line Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=09h)

Default: 00h

Attributes: Read/Write

Bit	Description
7	**Double Scanning Enable.** 0 = Disable. When disabled, the clock to the row scan counter is equal to the horizontal scan rate. This is the normal setting for many of the standard VGA modes. 1 = Enable. When enabled, the clock to the row scan counter is divided by 2. This is normally used to allow CGA-compatible modes that have only 200 scan lines of active video data to be displayed as 400 scan lines (each scan line is displayed twice).
6	**Line Compare Bit 9.** This bit provides the most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 4 of the Overflow Register (CR07) supplies the second most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.
5	**Vertical Blanking Start Bit 9.** The vertical blanking start is a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by this bit and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal to the number of scan line from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.

Bit	Description
4:0	**Starting Row Scan Count.** This field provides all 5 bits of a 5-bit value that specifies the number of scan lines in a horizontal row of text. This value should be programmed to be equal to the number of scan lines in a horizontal row of text, minus 1.

CR0A - Text Cursor Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ah)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5	**Text Cursor Off.** This text cursor exists only in text modes, so this register is entirely ignored in graphics modes. 0 = Enables the text cursor. 1 = Disables the text cursor.
4:0	**Text Cursor Start.** This field specifies which horizontal line of pixels in a character box is to be used to display the first horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the first horizontal line of pixels on which the cursor is to be shown.

CR0B - Text Cursor End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Bh)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Text Cursor Skew.** This field specifies the degree to which the start and end of each horizontal line of pixels making up the cursor is delayed to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = No delay 01 = Delayed by 1 character clock 10 = Delayed by 2 character clocks 11 = Delayed by 3 character clocks
4:0	**Text Cursor End.** This field specifies which horizontal line of pixels in a character box is to be used to display the last horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the last horizontal line of pixels on which the cursor is to be shown.

CR0C - Start Address High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ch)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [15:8].**This register provides either bits 15 through 8 of a 16-bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes, the start address is specified with a 16-bit value. The eight bits of this register provide the eight most significant bits of this value, while the eight bits of the Start Address Low Register (CR0D) provide the eight least significant bits.

CR0D - Start Address Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Dh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [7:0]** This register provides either bits 7 through 0 of a 16 bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes the start address is specified with a 16-bit value. The eight bits of the Start Address High Register (CR0C) provide the eight most significant bits of this value, while the eight bits of this register provide the eight least significant bits.

CR0E - Text Cursor Location High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Eh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [15:8].** This field provides the 8 most significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bit 7:0 of the Text Cursor Location Low Register (CR0F) provide the 8 least significant bits.

CR0F - Text Cursor Location Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Fh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [7:0].** This field provides the 8 least significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bits 7:0 of the Text Cursor Location High Register (CR0E) provide the 8 most significant bits.

CR10 - Vertical Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=10h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Sync Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area of a screen. In standard VGA modes, this value is described in 10 bits with bits [7,2] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should equal the vertical sync start in terms of the number of scan lines from the beginning of the active display area to the beginning of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

CR11 - Vertical Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=11h)

Default: 0U00 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Protect Registers [0:7].** The ability to write to Bit 4 of the Overflow Register (CR07) is not affected by this bit (i.e., bit 4 of the Overflow Register is always writeable). 0 = Enable writes to registers CR[00:07]. (default) 1 = Disable writes to registers CR[00:07].
6	**Reserved.** In the VGA standard, this bit was used to switch between 3 and 5 frame buffer refresh cycles during the time required to draw each horizontal line.
5	**Vertical Interrupt Enable.** This bit is reserved for compatibility only. While this bit may be written or read, it's value will have no effect. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. Bit 7 of Input Status Register 0 (ST00) originally indicated the status of the vertical retrace interrupt. 0 = Enable the generation of an interrupt at the beginning of each vertical retrace period. 1 = Disable the generation of an interrupt at the beginning of each vertical retrace period.
4	**Vertical Interrupt Clear.** This is reserved for compatibility only. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. 0 = Setting this bit to 0 clears a pending vertical retrace interrupt. This bit must be set back to 1 to enable the generation of another vertical retrace interrupt.
3:0	**Vertical Sync End.** This 4-bit field provides a 4-bit value that specifies the end of the vertical sync pulse relative to its beginning. This 4-bit value should be set to the least significant 4 bits of the result of adding the length of the vertical sync pulse in terms of the number of scan lines that occur within the length of the vertical sync pulse to the value that specifies the beginning of the vertical sync pulse (see the description of the Vertical Sync Start Register for more details).

CR12 - Vertical Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=12h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Display Enable End Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the number of the last scan line within the active display area. In standard VGA modes, this value is described in 10 bits with bits [6,1] of the Overflow Register (CR07) supplying the two most significant bits. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area,minus 1.

CR13 - Offset Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=13h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Offset Bits [7:0].** This register provides either all 8 bits of an 8-bit value that specifies the number of words or DWords of frame buffer memory occupied by each horizontal row of characters. Whether this value is interpreted as the number of words or DWords is determined by the settings of the bits in the Clocking Mode Register (SR01). In standard VGA modes, the offset is described with an 8-bit value, all the bits of which are provided by this register. This 8-bit value should be programmed to be equal to either the number of words or DWords (depending on the setting of the bits in the Clocking Mode Register, SR01) of frame buffer memory that is occupied by each horizontal row of characters.

CR14 - Underline Location Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=14h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6	**DWord Mode.** 0 = Frame buffer addresses are interpreted by the frame buffer address decoder as being either byte addresses or word addresses, depending on the setting of bit 6 of the CRT Mode Control Register (CR17). 1 = Frame buffer addresses are interpreted by the frame buffer address decoder as being DWord addresses, regardless of the setting of bit 6 of the CRT Mode Control Register (CR17). This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to select how frame buffer addresses from the CPU are interpreted by the frame buffer address decoder as shown below: **CR14[6] / CR17[6] / Addressing Mode** 0 / 0 / Word Mode 0 / 1 / Byte Mode 1 / 0 / DWord Mode 1 / 1 / DWord Mode
5	**Count By 4.** 0 = The memory address counter is incremented either every character clock or every other character clock, depending upon the setting of bit 3 of the CRT Mode Control Register. 1 = The memory address counter is incremented either every 4 character clocks or every 2 character clocks, depending upon the setting of bit 3 of the CRT Mode Control Register. . This is used in mode x13 to allow for using all four planes. This bit is used in conjunction with bit 3 of the CRT Mode Control Register (CR17) to select the number of character clocks are required to cause the memory address counter to be incremented as shown, below: **CR14[5] / CR17[3] / Addressing Incrementing Interval** 0 / 0 / every character clock 0 / 1 / every 2 character clocks 1 / 0 / every 4 character clocks 1 / 1 / every 2 character clocks

Bit	Description
4:0	**Underline Location.** This field specifies which horizontal line of pixels in a character box is to be used to display a character underline in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the horizontal line on which the character underline mark is to be shown.

CR15 - Vertical Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=15h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Blanking Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area of the screen. In standard VGA modes, the vertical blanking start is specified with a 10-bit value. The most and second-most significant bits of this value are supplied by bit 5 of the Maximum Scan Line Register (CR09) and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal the number of scan lines from the beginning of the active display area to the beginning of the vertical blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which vertical blanking begins.

CR16 - Vertical Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=16h)

Default: Undefined

Attributes: Read/Write

This register provides a 8-bit value that specifies the end of the vertical blanking period relative to its beginning.

Bit	Description
7:0	**Vertical Blanking End Bits [7:0].** This 8-bit value should be set equal to the least significant 8 bits of the result of adding the length of the vertical blanking period in terms of the number of scan lines that occur within the length of the vertical blanking period to the value that specifies the beginning of the vertical blanking period (see the description of the Vertical Blanking Start Register for details).

CR17 - CRT Mode Control

I/O (and Memory Offset) Address: 3B5h/3D5h (index=17h)

Default: 0UU0 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**CRT Controller Reset.** This bit has no effect except in native VGA modes (non-centered). 0 = Forces horizontal and vertical sync signals to be inactive. No other registers or outputs are affected. 1 = Permits normal operation.
6	**Word Mode or Byte Mode.** 0 = The memory address counter's output bits are shifted by 1 bit position before being passed on to the frame buffer address decoder such that they are made into word-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. 1 = The memory address counter's output bits remain unshifted before being passed on to the frame buffer address decoder such that they remain byte-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to control how frame buffer addresses from the memory address counter are interpreted by the frame buffer address decoder as shown below: CR14[6] CR17[6] **Address Mode** 0 0 Word Mode - Addresses from the memory address counter are shifted once to become word-aligned 0 1 Byte Mode - Addresses from the memory address counter are not shifted 1 0 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned 1 1 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned
5	**Address Wrap.** This bit is only effective when word mode is made active by setting bit 6 in both the Underline Location Register and this register to 0. 0 = Wrap frame buffer address at 16 KB. This is used in CGA-compatible modes. 1 = No wrapping of frame buffer addresses.

Bit	Description
4	**Reserved.** Read as 0.
3	**Count By 2.** This bit is used in conjunction with bit 5 of the Underline Location Register (CR14) to select the number of character clocks are required to cause the memory address counter to be incremented. 0 = The memory address counter is incremented either every character clock or every 4 character clocks, depending upon the setting of bit 5 of the Underline Location Register. 1 = The memory address counter is incremented either every other clock. CR14[5] CR17[3] Address Incrementing interval 0 0 every character clock 0 1 every 2 character clocks 1 0 every 4 character clocks 1 1 every 2 character clocks
2	**Horizontal Retrace Select.** This bit provides a way of effectively doubling the vertical resolution by allowing the vertical timing counter to be clocked by the horizontal retrace clock divided by 2 (usually, it would be undivided). 0 = The vertical timing counter is clocked by the horizontal retrace clock. 1 = The vertical timing counter is clocked by the horizontal retrace clock divided by 2.
1	**Select Row Scan Counter.** 0 = A substitution takes place, where bit 14 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 1 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.
0	**Compatibility Mode Support.** 0 = A substitution takes place, where bit 13 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 0 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.

The following tables show the possible ways in which the address bits from the memory address counter can be shifted and/or reorganized before being presented to the frame buffer address decoder. First, the

address bits generated by the memory address counter are reorganized, if need be, to accommodate byte, word or DWord modes. The resulting reorganized outputs (MAOut15-MAOut0) from the memory address counter may also be further manipulated with the substitution of bits from the row scan counter (RSOut1 and RSOut0) before finally being presented to the input bits of the frame buffer address decoder (FBIn15-FBIn0).

Memory Address Counter Address Bits [15:0]

	Byte Mode CR14 bit 6=0 CR17 bit 6=1 CR17 bit 5=X	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=1	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=0	DWord Mode CR14 bit 6=1 CR17 bit 6=X CR17 bit 5=X
MAOut0	0	15	13	12
MAOut1	1	0	0	13
MAOut2	2	1	1	0
MAOut3	3	2	2	1
MAOut4	4	3	3	2
MAOut5	5	4	4	3
MAOut6	6	5	5	4
MAOut7	7	6	6	5
MAOut8	8	7	7	6
MAOut9	9	8	8	7
MAOut10	10	9	9	8
MAOut11	11	10	10	9
MAOut12	12	11	11	10
MAOut13	13	12	12	11
MAOut14	14	13	13	12
MAOut15	15	14	14	13

X = Don't Care

Frame Buffer Address Decoder

	CR17 bit 1=1 CR17 bit 0=1	CR17 bit 1=1 CR17 bit 0=0	CR17 bit 1=0 CR17 bit 0=1	CR17 bit 1=0 CR17 bit 0=0
FBIn0	MAOut0	MAOut0	MAOut0	MAOut0
FBIn1	MAOut1	MAOut1	MAOut1	MAOut1
FBIn2	MAOut2	MAOut2	MAOut2	MAOut2
FBIn3	MAOut3	MAOut3	MAOut3	MAOut3
FBIn4	MAOut4	MAOut4	MAOut4	MAOut4
FBIn5	MAOut5	MAOut5	MAOut5	MAOut5
FBIn6	MAOut6	MAOut6	MAOut6	MAOut6
FBIn7	MAOut7	MAOut7	MAOut7	MAOut7
FBIn8	MAOut8	MAOut8	MAOut8	MAOut8
FBIn9	MAOut9	MAOut9	MAOut9	MAOut9
FBIn10	MAOut10	MAOut10	MAOut10	MAOut10
FBIn11	MAOut11	MAOut11	MAOut11	MAOut11
FBIn12	MAOut12	MAOut12	MAOut12	MAOut12
FBIn13	MAOut13	MAOut13	RSOut0	RSOut0
FBIn14	MAOut14	RSOut1	MAOut14	RSOut1
FBIn15	MAOut15	MAOut15	MAOut15	MAOut15

CR18 - Line Compare Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=18h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Line Compare Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bit 4 of the Overflow Register (CR07) supplies the second most significant bit. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. (This register is only used in split screening modes, and this is not a problem because split screening is not actually used for extended modes. As a result, there is no benefit to extending the existing overflow bits for higher resolutions.) When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

CR22 - Memory Read Latch Data Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=22h)

Default: 00h

Attributes: Read Only

Bit	Description
7:0	**Memory Read Latch Data.** This field provides the value currently stored in 1 of the four memory read latches. Bits 1 and 0 of the Read Map Select Register (GR04) select which of the four memory read latches may be read via this register.

CR24 - Toggle State of Attribute Controller Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=24h)

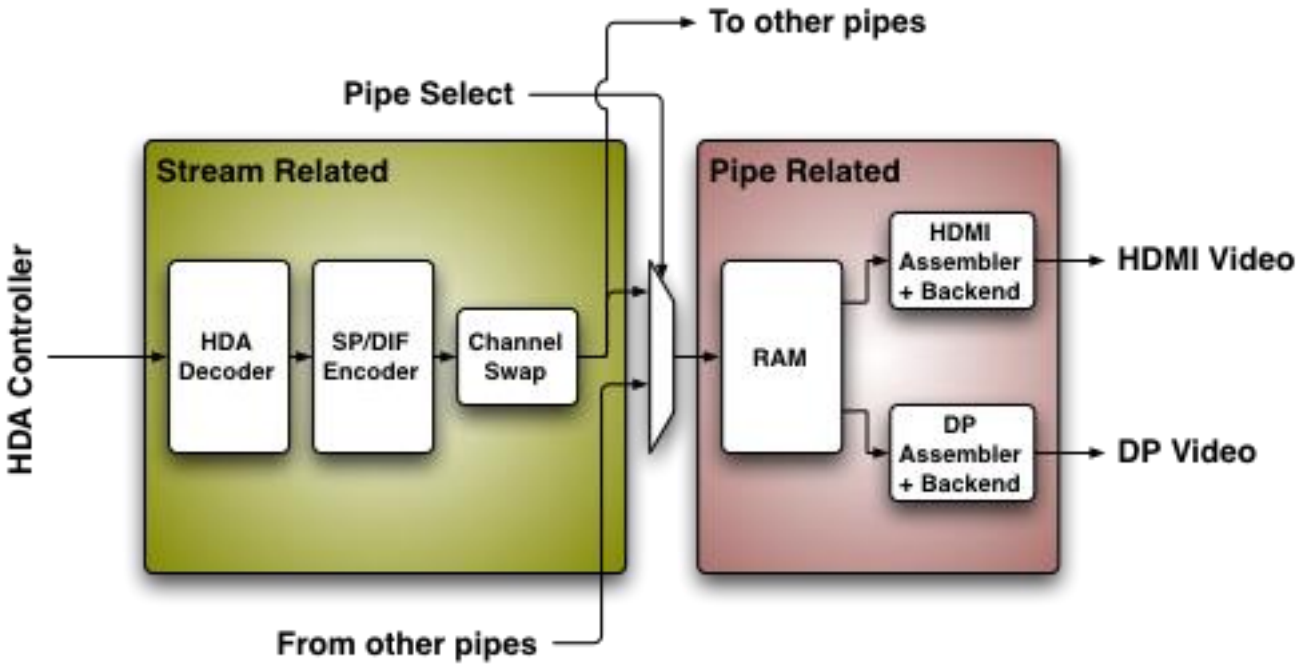
Default: 00h

Attributes: Read Only

Bit	Description
7	**Toggle Status.** Indicates where the last write to attribute register was to: 0 = index port 1 = data port
6:0	**Reserved.** Read as 0.

Display Audio Codec Verbs

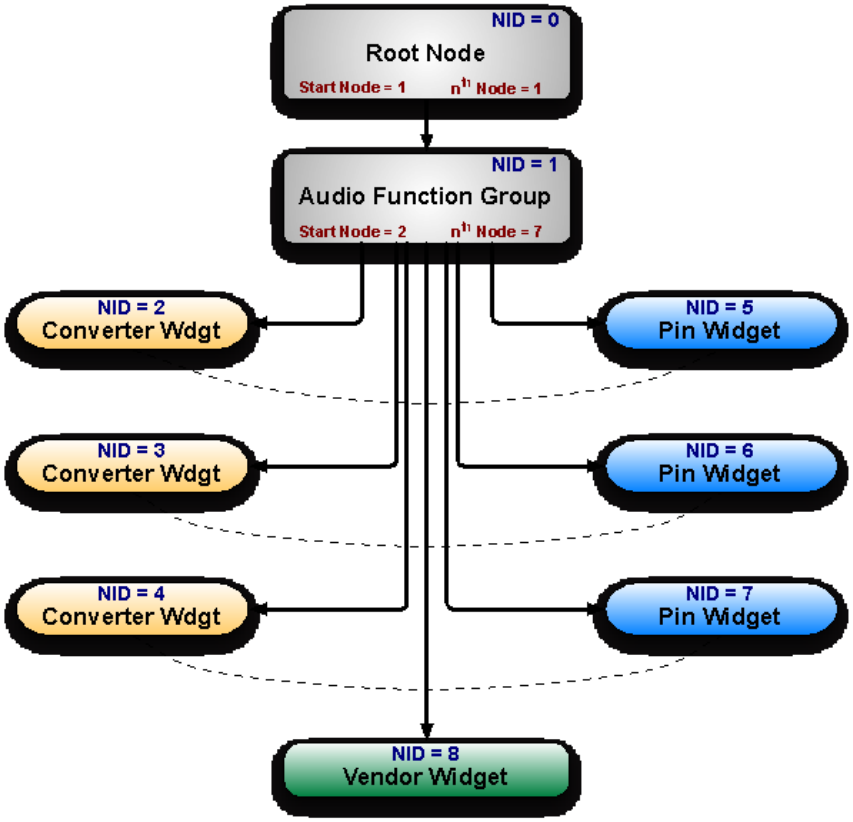
Block Diagram



Codec Node Hierarchy

The diagram below shows the hierarchy of the internal codec. The codec is presented as a single codec with multiple endpoints. By operating as a single codec, only one driver needs to be loaded on the system.

Inside the codec are three "converter widgets" and three "pin widgets", responsible for taking data from HD Audio DMA engines and placing into an HDMI/DP stream. Each pin widget has a 1-1 connection to a converter widget (as indicated by the dotted lines in the diagram).



Programming

Programming of the codec is performed by "verbs" as described in the HD Audio specification. These verbs travel over the internal HD Audio link at a rate of 1 verb per frame. A verb can either come from the CORB, with responses using the RIRB, or using an immediate command and response mechanism (ICR). Device 2 contains its own copy of an ICR mechanism as a back-door into the audio codec.

Verb Support

Verb ID		Verb Name/Description	Node ID							
Set	Get		01h	02h	03h	04h	05h	06h	07h	08h
2h	Ah	Stream Descriptor Format		Y	Y	Y				
3h	Bh	Set Amplifier Mute					Y	Y	Y	
-	F00h	Get Parameters	Y	Y	Y	Y	Y	Y	Y	Y
701h	F01h	Connection Select Control					Y	Y	Y	
-	F02h	Connection List Entry					Y	Y	Y	
705h	F05h	Power State		Y	Y	Y	Y	Y	Y	
706h	F06h	Channel and Stream ID		Y	Y	Y				
707h	F07h	Pin Widget Control					Y	Y	Y	
708h	F08h	Unsolicited Response Enable					Y	Y	Y	Y
-	F09h	Pin Sense					Y	Y	Y	
-	F0Dh	Digital Converter		Y	Y	Y				
70Dh	-	Digital Converter 1		Y	Y	Y				
70Eh	-	Digital Converter 2		Y	Y	Y				
-	F1Ch	Configuration Default					Y	Y	Y	
71Ch	-	Configuration Default Byte 0					Y	Y	Y	
71Dh	-	Configuration Default Byte 1					Y	Y	Y	
71Eh	-	Configuration Default Byte 2					Y	Y	Y	
71Fh	-	Configuration Default Byte 3					Y	Y	Y	
-	F20h	Subsystem ID	Y							
-	F21h	Subsystem ID	Y							
-	F22h	Subsystem ID	Y							
-	F23h	Subsystem ID	Y							
720h	-	Subsystem ID[7: 0]	Y							
721h	-	Subsystem ID[15: 8]	Y							
722h	-	Subsystem ID[23:16]	Y							
723h	-	Subsystem ID[31:24]	Y							
72Dh	F2Dh	Converter Channel Count		Y	Y	Y				
-	F2Eh	HDMI/DP Info Size					Y	Y	Y	

Verb ID		Verb Name/Description	Node ID							
Set	Get		01h	02h	03h	04h	05h	06h	07h	08h
730h	F30h	HDMI Info Index					Y	Y	Y	
731h	F31h	HDMI Info Data					Y	Y	Y	
732h	F32h	HDMI Info Transmit Control					Y	Y	Y	
734h	F34h	Converter Channel Map					Y	Y	Y	
735h	F35h	Device Select					Y	Y	Y	
-	F36h	Display Device List Entry					Y	Y	Y	
73Ch	73Ch	DisplayPort Stream ID					Y	Y	Y	
73Eh	-	Digital Converter 3		Y	Y	Y				
73Fh	-	Digital Converter 4		Y	Y	Y				
-	F80h	HDMI / DP Status								Y
781h	F81h	HDMI Vendor Verb								Y
782h	-	GTC Capture Trigger								Y
-	F83h	Captured Wall Clock Value								Y
-	F84h	Captured GTC Value								Y
-	F85h	Get GTC Offset Value								Y
785h	-	Set GTC Offset Value[7: 0]								Y
786h	-	Set GTC Offset Value[15: 8]								Y
787h	-	Set GTC Offset Value[23:16]								Y
788h	-	Set GTC Offset Value[31:24]								Y
789h	F89h	Converter Channel Count								Y

Parameter Support

Param ID	Parameter Name	Node ID								
		00h	01h	02h	03h	04h	05h	06h	07h	08h
00h	Vendor ID	Y								
02h	Revision ID	Y								
04h	Subordinate Node Count	Y	Y							
05h	Function Group Type		Y							
08h	Audio Function Group Capabilities									
09h	Audio Widget Capabilities			Y	Y	Y	Y	Y	Y	Y
0Ah	Sample Size, Rate CAPs			Y	Y	Y				
0Bh	Stream Formats			Y	Y	Y				
0Ch	Pin Capabilities						Y	Y	Y	
0Dh	Input Amp Capabilities									
0Eh	Connection List Length						Y	Y	Y	
0Fh	Supported Power States		Y							
10h	Processing Capabilities									
11h	GPIO Count									
12h	Output Amp Capabilities						Y	Y	Y	
13h	Volume Knob Capabilities									
15h	Device List Length						Y	Y	Y	

Node ID 00h Root Node Verbs

The root node only contains a single verb - the "Get Parameters" verb at F00h.

F00h - Get Parameters

Parameter	Symbol	Register Name
00h	PARAM_VID	Vendor ID
02h	PARAM_RID	Revision ID
04h	PARAM_SNC	Subordinate Node Count

Parameter 00h: VID - Vendor ID

Bit	Reset	Description
31:16	8086h	**Vendor ID (VID):** Indicates the 16-bit Vendor ID values used to identify the codec to the PnP subsystem.
15:00	2809h	**Device ID (DID):** Indicates the 16-bit Device ID values used to identify the codec to the PnP subsystem. It is 280Bh for KBL

Parameter 02h: RID - Revision ID

Bit	Reset	Description
31:24	*0*	*Reserved*
23:20	1h	**Major Revision (MJR):** Indicates the major revision number (left of the decimal) of the High Definition Audio Specification to which the codec is fully compliant.
19:16	0h	**Minor Revision (MNR):** Indicates the minor revision number (right of the decimal) or "dot number" of the High Definition Audio Specification to which the codec is fully compliant.
15:08	00h	**Revision ID (RID):** Indicates the vendor's revision number for this given Device ID.
07:00	00h	**Stepping ID (SID):** Indicates optional vendor stepping number within the revision.

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	0h	**Starting Node Number (SNN):** Indicates the first sub-node's ID is 01h.
15:08	00h	Reserved
07:00	01h	**Total Number of Nodes (TNN):** Indicates one sub-node

F37h GET CCF - Get Current Clock Frequency

Bits	Default	Description
31:6	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz. Reserved for Display Codec
04	1	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz. Reserved for Display Codec
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz. Reserved for Display Codec
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz. Reserved for Display Codec

Node ID 01h Audio Function Group Verbs

Set Verb	Get Verb	Symbol	Name
-	F00h	GET PARAM	Get Parameters
705h	F05h	SET_PS / GET_PS	Set Power State
-	F20h	GET_SSID	Get Subsystem ID
720h	F20h	SET_SSID0	Set/Get Subsystem ID
721h	F21h	SET_SSID1	Set/Get Subsystem ID
722h	F22h	SET_SSID2	Set//Get Subsystem ID
723h	F23h	SET_SSID3	Set/Get Subsystem ID
724h	F24h	SET CCF	Set/Get Current Clock Frequency

F00h Get Parameters

Parameter	Symbol	Register Name
04h	PARAM_SNC	Subordinate Node Count
05h	PARAM_FGT	Function Group Type
08h	PARAM_FGC	Function Group Capabilities
0Fh	PARAM SPS	Supported Power States

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	02h	**Start Node Number (SNN):** Indicates the start node number of widget or functional nodes in the Functional Group.
15:08	*0*	*Reserved*
07:00	07h	**Total Number of Nodes (TNN):** Indicates 7 widgets in the Functional Group. (HDMI/DP converters (3) + HDMI/DP pins (3) + Vendor Defined Widget (1)).

Parameter 05h: PARAM_FGT - Function Group Type

Bit	Reset	Description
31:09	*0*	*Reserved*
08	0	**Unsolicited Capable (UC):** Not capable of generating an unsolicited response.
07:00	01h	**Node Type (NT):** Indicates Audio Function Group.

Parameter 08h: PARAM_FGC - Function Group Capability

Bit	Reset	Description
31:04	*0*	*Reserved*
03:00	00h	**Output Delay (OD)**Output Delay.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30	1	**Clock Stop (CS):** Indicates support for D3 when clock is stopped.
29:04	*0*	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

Parameter 16h: PARAM_A2CAP - Azalia 2 Capabilities

Bits	Reset	Description
31:17	*0*	*Reserved*
16	0	**Independent Codec Clock (ICC):** When set, this indicates that the codec generates its own clock, which may drift from the link clock. When cleared, the codec's clock is locked to the link clock.
15:06	*0*	*Reserved*
05	*0*	*Reserved for 192 MHz support.*
04	1	**96MHz Supported (S96):** Indicates 96 MHz clock is supported.
03	1	**48MHz Supported (S48):** Indicates 48 MHz clock is supported.This bit must always be set.
02	0	**24MHz Supported (S24):** Indicates 24 MHz clock is supported. Reserved for Display Codec
01	0	**12MHz Supported (S12):** Indicates 12 MHz clock is supported. Reserved for Display Codec
00	0	**6MHz Supported (S6)**Indicates 6 MHz clock is supported. Reserved for Display Codec

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	1	**Clock Stop OK (CSOK):** Clock stopping in D3 is OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

F20h GET SSID - Get Subsystem ID0

Bits	Reset	Description
31:00	80860101h	**Subsystem ID (SSID):** Reports the sub-system ID set via SET_SSIDx verbs.

720h SET SSID0 - Set Subsystem ID0

Bits	Description
07:00	Subsystem ID Bits [7:0]

721h SET SSID1 - Set Subsystem ID1

Bits	Description
07:00	Subsystem ID Bits [15:8]

722h SET SSID2 - Set Subsystem ID2

Bits	Description
07:00	Subsystem ID Bits [23:16]

723h SET SSID3 - Set Subsystem ID3

Bits	Description
07:00	Subsystem ID Bits [31:24]

724h SET CCF - Set Current Clock Frequency

Bits	Description
07:06	Reserved
05	**Set Clock 192 MHz (S192)**: Set clock to 192 MHz.
04	**Set Clock 96 MHz (S96)**: Set clock to 96 MHz
03	**Set Clock 48 MHz (S48)**: Set clock to 48 MHz
02	**Set Clock 24 MHz (S24)**: Set clock to 24 MHz
01	**Set Clock 12 MHz (S12)**: Set clock to 12 MHz
00	**Set Clock 6 MHz (S6)**: Set clock to 6 MHz

F24h GET CCF - Get Current Clock Frequency

Bits	Bits	Description
31:06	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz.
04	0	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz.
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz.
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz.

7FFh SET Function Group Reset

Bits	Reset	Description
07:00	00h	The Function Reset command causes the functional unit, and all widgets associated with the functional unit, to return to their power-on reset values. Note that some controls such as the Configuration Default controls should not be reset with this command. It is also possible that certain other controls, such as Caller-ID, should not be reset. This command does not affect the Link interface logic, which must be reset with the link RST# signal. Therefore, a codec must not initiate a Status Change request on the link.

Node ID 02h 03h 04h Audio Output Convertor Widget Verbs

Verb	Symbol	Verb Name
2h	SET_SDF	Set Stream Descriptor Format
Ah	GET_SDF	Get Stream Descriptor Format
F00h	GET_PARAM	Get Parameters
705h	SET_PS	Set Power State
F05h	GET_PS	Get Power State
706h	SET_CSID	Set Channel and Stream ID
F06h	GET_CSID	Get Channel and Stream ID
F0Dh	SET_DC1	Get Digital Converter
70Dh	SET_DC1	Set Digital Converter 1

Verb	Symbol	Verb Name
70Eh	SET_DC2	Set Digital Converter 2
73Eh	SET_DC3	Set Digital Converter 3
73Fh	SET_DC4	Set Digital Converter 4
72Dh	SET_CCC	Set Converter Channel Count
F2Dh	GET_CCC	Get Converter Channel Count

2hAh SETGET_SDF - SetGET Stream Descriptor Format

Bits	Reset	Description
31:15	0	Reserved
14	0	**Sample Base Rate (SBR):**
13:11	000	**Sample Base Rate Multiplier (SBRM):**
10:08	000	**Sample Base Rate Divisor (SBRD):**
07	0	Reserved
06:04	011	**Bits / Sample (BPS):** • 001b: Data is packed in memory in 16 bit containers on 16 bit boundaries • 010b: Data is packed in memory in 20 bit containers on 32 bit boundaries • 011b: Data is packed in memory in 24 bit containers on 24 bit boundaries • 100b: Data is packed in memory in 32 bit containers on 32 bit boundaries All other bit combinations reserved
03:00	1h	**# Channels in Stream (NCS):** 2 channels in each frame

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ah	PARAM_PSB	Parameter Sizes and Bit Rates
0Bh	PARAM_SF	Stream Formats
0Fh	PARAM_SPS	Power Supported States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	0h	**Widget Type (TYPE):** Indicates this is an audio output widget
19:16		Sample Delay in Widget (DELAY):
15:13	011	**Channel Count Extension (CCE):** These three bits, combined with STRO, indicate that there are 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	0	**Connection List (CL):** Indicates no connection list
07	0	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	1	**Format Override (FO):** Indicates support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier support.
02	0	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ah: PSB - PCM Sizes and Bit Rates

Bits	Reset	Description
31:21	0	*Reserved*
20	1	**32-bit Support (B32):** Indicates 32-bit samples supported
19	1	**24-bit Support (B24):** Indicates 24-bit samples supported
18	0	**20-bit Support (B20):** Indicates 20-bit samples supported
17	1	**16-bit Support (B16):** Indicates 16-bit samples supported
16	0	**8-bit Support (B8):** Indicates 8-bit samples not supported
15:12	0	*Reserved*
11	0	**384 kHz Support (R12):** Indicates 384 kHz not supported
10	1	**192 kHz Support (R11):** Indicates 192 kHz supported
09	1	**176.4 kHz Support (R10):** Indicates 176.4 kHz supported
08	1	**96 kHz Support (R9):** Indicates 96 kHz supported
07	1	**88.2 kHz Support (R8):** Indicates 88.2 kHz supported
06	1	**48 kHz Support (R7):** Indicates 48 kHz supported
05	1	**44.1 kHz Support (R6):** Indicates 44.1 kHz supported
04	1	**32 kHz Support (R5):** Indicates 32 kHz supported
03	0	**22.05 kHz Support (R4):** Indicates 22.05 kHz not supported
02	0	**16 kHz Support (R3):** Indicates 16 kHz not supported
01	0	**11.025 kHz Support (R2):** Indicates 11.025 kHz not supported
00	0	**8 kHz Support (R1):** Indicates 8 kHz not supported

Parameter 0Bh: SF - Stream Formats

Bits	Reset	Description
31:03	0	*Reserved*
02	1	**AC3 Support (AC3):** Indicates AC3 stream format is supported
01	0	**Float32 Support (F32):** Indicates float32 stream format not supported
00	1	**PCM Support (PCM):** Indicates PCM format is supported.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** ???
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK

Bits	Reset	Description
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

706hF06h GETSET_CSID - GetSet Channel and Stream ID

Bits	Reset	Description
07:04	0h	**Stream ID (SID):** Link stream used by the converter for data output.
03:00	0h	**Lowest Channel Number (LCN):** Lowest channel used by the converter.

Digital Converter Verbs

F0Dh: GET_DC - Get Digital Converter

Bits	Reset	Description
31:24	0	Reserved
23	1	**Keep Alive (KA):** See SET_DC3.KA
22:20	0	Reserved
19:16	0h	**IEC Coding Type (ICT):** See SET_DC3.ICT
15	0	Reserved
14:08	00h	**Category Code (CC):** See SET_DC1.CC
07	0	**Level (LVL):** See SET_DC1.LVL
06	0	**Professional (PRO):** See SET_DC1.PRO
05	0	**Audio is not PCM (AUDIO):** See SET_DC1.AUDIO
04	0	**Copyright (COPY):** See SET_DC1.COPY
03	0	**Pre-emphasis (PRE):** See SET_DC1.PRE

Bits	Reset	Description
02	0	**Validity Configuration (VCFG):** See SET_DC1.VCFG
01	0	**Validity (V):** See SET_DC1.V
00	1	**Digital Enable (DIGEN):** See SET_DC1.DIGEN

70Dh: SET_DC1 - Set Digital Converter 1

Bits	Description
07	**Level (LVL):** S/PDIF IEC Generation Level.
06	**Professional (PRO):** When set, indicates professional use of channel.
05	**Audio is not PCM (AUDIO):** When set, data is non-PCM format.
04	**Copyright (COPY):** When set, copyright asserted.
03	**Pre-emphasis (PRE):** When set, enables filter pre-emphasis.
02	**Validity Configuration (VCFG):** Determines S/PDIF transmitter behavior when data is not being transmitted.
01	**Validity (V):** Affects the validity flag transmitted in each sub-frame, and enables S/PDIF transmitter to maintain connection during error or mute conditions.
00	**Digital Enable (DIGEN):** When set, enables digital content

70Eh: Digital Converter 2

Bits	Description
07	Reserved
06:00	**Category Code (CC):** S/PDIF IEC Category Code.

73Eh: Digital Converter 3

Bits	Description
07	Keep Alive
06:04	Reserved
03:00	IEC Coding Type

73Fh: Digital Converter 4

Bits	Description
07:00	Reserved

72DhF2Dh GETSET_CCC - GetSet Converter Channel Count

Bits	Reset	Description
07:04	0	Reserved
03:00	0000	Converter Channel Count 1 (0th order)

Node ID 05h 06h 07h Pin Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
3h	-	SET_AM	Set Amplifier Mute
-	Bh	GET_AM	Get Amplifier Mute
-	F00h	-	Get Parameters
701h	F01h	SET_CSC / GET_CSC	Set/Get Connection Select Control
-	F02h	-	Get Connection List Entry
705h	F05h	SET_PS / GET_PS	Set/Get Power State
707h	F07h	SET_PWC / GET_PWC	Set/Get Pin Widget Control
708h	F08h	SET_UE / GET_UE	Set/Get Unsolicited Response Enable
-	F09h	-	Get Pin Sense
71Ch	-	SET_CD0	Set Configuration Default Byte 0
71Dh	-	SET_CD1	Set Configuration Default Byte 1
71Eh	-	SET_CD2	Set Configuration Default Byte 2
71Fh	-	SET_CD3	Set Configuration Default Byte 3
-	F1Ch	GET_CD	Get Configuration Default
-	F2Eh	GET_HDIS	Get HDMI/DP Info Size
730h	F30h	SET_HII / GET_HII	Set/Get HDMI Info Index
731h	F31h	SET_HID / GET_HID	Set/Get HDMI Info Data
732h	F32h	SET_HITC / GET_HITC	Set/Get HDMI Info Transmit Control
733h	F33h	SET_PC / GET_PC	Set/Get Protection Control
734h	F34h	SET_CCM / GET_CCM	Set/Get Converter Channel Map
735h	F35h	SET_DS / GET_DS	Set/Get Device Select
-	F36h	GET_DDLE	Get Display Device List Entry
73Ch	F3Ch	SET_DPID / GET_DPID	Set/Get DisplayPort Stream ID

3h SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

B8h GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	0	*Reserved*
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
15h	PARAM_DLL	Device List Length
0Fh	PARAM_SPS	Supported Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	0	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio

Bits	Reset	Description
26	0	*Reserved*
25	1	**Multi Stream Capable (MSC):** Indicates support for DisplayPort Multistream. Will be zero in vanilla mode.
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	0	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI
06:05	0	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	0	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	0	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	0	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 15h: DLL - Device List Length

Bits	Reset	Description
31:06	0	*Reserved*
05:00	00h	**Length (LEN):** Indicates no devices

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

701hF01h SETGET_CSC - SetGet Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC):**

F02h GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:08	0	Reserved
07:00	Varies	**Connection List Entry (CLE):** 02h for NodeID 05h, 03h for NodeID 06h, and 04h for NodeID 07h.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707hF07h SETGET_PWC - SetGet Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT):**

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG):**

F09h GET_PS - Get Pin Sense

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV):**
29	0	**Inactive (INA):**
28:00	28:00	Reserved

71Ch SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	Default Association (DA):
03:00	Sequence (SEQ):

71Dh SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	Color (COL):
03:00	Miscellaneous (MISC):

71Eh SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	Default Device (DD):
03:00	Connection Type (CT):

71Fh SET_CD3 - Set Configuration Default Byte 3

Bits	Description
07:06	**Port Connectivity (PC):** External connectivity of the pin complex. • 00 = Connected to jack • 01 = No physical connection • 10 = Fixed function device (integrated speaker, mic, etc.) • 11 = Both a jack and internal connection
05:00	**Location (LOC):**

Bits	Description

	Bits 3:0										
Bits 5:4	0h: N/A	1h: Rear	2h: Front	3h: Left	4h: Right	5h: Top	6h: Bottom	7h: Special	8h: Special	9h: Special	Ah-Fh Reserved
00: External	Y	Y	Y	Y	Y	Y	Y	Y	Y		
01: Internal	Y							Y	Y	Y	
10: Separate Chassis	Y	Y	Y	Y	Y	Y	Y				
11: Other	Y						Y	Y	Y		

F1Ch GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Eh HDMIDP Info Size

Bits	Reset	Description
31:08	0	Reset
07:00	Varies	**Size (SZ):** Indexes 0 - 3 return 1Eh, index 1000 returns 53h, others reserved.

F2Fh Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

730hF30h SETGET_HII - SetGet HDMI Info Index

Bits	Reset	Description
07:05	000	**Infoframe Packet Index (IPI):** Value / Name / Value / Name 000 / Audio / 011 / GP3 001 / GP / 100 / GP4 010 / GP2 / Others / Reserved
04:00	00h	**Byte Offset Index Pointer (BOI):**

731hF31h SETGET_HID - SetGet HDMI Info Data

Bits	Reset	Description
07:00	00h	**Data (DATA):** Data at current index pointed to from SET_HII verb.

732hF32h SETGET_HITC - SetGet HDMI Info Transmit Control

Bits	Reset	Description
07:06	00	**InfoFrame Control Current Indexed Frame (IFCCIF):** • 00 = Disable Transmit • 01 = Reserved • 10 = Transmit Once • 11 = Best Effort
05:00	0	*Reserved*

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734hF34h SETGET_CCM - GetSet Converter Channel Map

Bits	Reset	Description
07:04	0h	Converter Channel (CC):
03:00	0h	Slot (SN):

735h SET_DS - Set Device Select

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present)

F35h: GET_DS - Get Device Select

Bits	Description
31:12	**Reserved:** Set to 0
11:06	**SinK Device ID:** Sink Device ID in the multi stream topology of the DP hierarchy. Device attached to Pipe A will have ID of "00000", PipeB will have "00001" and Pipe C will have "00010".
05:00	**Device (D):**Device Entry index currently set

F36h GET_DDLE - Get Display Device List Entry

Bits	Bits	Description
31:12	0	*Reserved*
11	0	**Reserved?**
10	0	**IA of Entry 2**
09	0	**ELDV of Entry 2**

Bits	Bits	Description
08	0	**PD of Entry 2**
07	0	**Reserved?**
06	0	**IA of Entry 1**
05	0	**ELDV of Entry 1**
04	0	**PD of Entry 1**
03	0	**Reserved?**
02	0	**IA of Entry 0**
01	0	**ELDV of Entry 0**
00	0	**PD of Entry 0**

73ChF3Ch SETGET_DPID - SetGet DisplayPort Stream ID

Bits	Reset	Description
07:03	00h	**Tag (TAG):** Represents the SSID that will go in the lower 5 bits of the SSID
02:00	000	**Index (IDX):** Pointer to program multiple SSID

Node ID 08h Intel Vendor Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
-	F00h	GET_PARAM	Get Parameters
-	F80h	GET_HDPS	Get HDMI/DP Status
781h	F81h	SET_HVV / GET_HVV	Set/Get iDisp Codec Vendor Verb
782h	-	SET_GTCT	Set GTC Trigger
-	F83h	GET_CWC	Get Captured Wall Clock
	F84h	GET_CGTC	Get Captured GTC Value
	F85h	GET_GOF	Get GTC Offset Value
785h	-	SET_GOF0	Set GTC Offset Value Byte 0
786h	-	SET_GOF1	Set GTC Offset Value Byte 1
787h	-	SET_GOF2	Set GTC Offset Value Byte 2
788h	-	SET_GOF3	Set GTC Offset Value Byte 3

Set Verb	Get Verb	Symbol	Verb Name
789h	F89h	SET_GDI / GET_GDI	Set/Get GTC Offset Device Index

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	Fh	**Widget Type (TYPE):** Indicates this is a vendor defined widget
19:00	0	Reserved

71Eh SET_GET_GFXMAILBOX - Set Get GFX MAILBOX Byte 2

71Eh: SET GFX MAILBOXM

Bits	Default	Description
07:00	00	Contents to be defined by GFX driver and audio driver

F1Eh: GET GFX MAILBOX

Bits	Default	Description
31:24	00h	Other values of 71F, verbs
23:16	00h	Contents to be defined by GFX driver and audio driver
15:00	0000h	Other values of 71D, 71C verb

728h SET CLOCK OFF - Set Clock Off Command

Bits	Description
07:00	Data is Irrelevant

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses from GFX MAIL BOX register writes are allowed.
06	Reserved
05:00	**Tag (TAG):** Tag for GFX Mail box register unsol responses.

781hF81h GETSET_VV - GetSet iDisp Codec Vendor Verb

Bits	Bits	Description
07:06	0	**Port Select:** This filed is programmed to select the port to be exposed to the inbox driver in the vanilla mode. Will not reset on double function group reset. 00 -> Port B 01 -> Port C 10 -> Port D 11 -> Reserved
05:04	0	**Reserved**
03	0	**Enable WFA:** When Set, WFA mode is enabled**.** Will not reset on double function group reset.
02	0	**Reserved**
01	0	**Enable DP1.2 Features (EDP12):** When set, DP1.2 features are enabled.Will not reset on double function group reset.
00	0	**Enable 3rd Pin and Converter Widget (E3P):** When set, the second and third pin and converter widget is enabled and can respond to HD Audio Verbs. When cleared, the second and third pin and converter widget is disabled and cannot respond to HD Audio verbs.Will not reset on double function group reset.

782h SET_GTCT - Set GTC Trigger

Bits	Bits	Description
07:00	0	**Any data:** The value of this field is irrelevant. The access of the SET causes a capture to occur.

F83h GET_CGTC - Get Captured GTC Value

Bits	Bits	Description
07	0	**GTC Value:** 32-bit GTC value captured on the SET_GTCT verb

F84h GET_CWC - Get Captured Wall Clock Value

Bits	Bits	Description
31:00	0	**Wall Clock Value:** 32-bit wall clock value captured on the SET_GTCT verb

F85h GET GOF - Get GTC Offset Value

Bits	Reset	Description
31:00	0h	**Value (VAL):** Reports the GTC Offset Value.

785h SET GOF0 - Set GTC Offset Value Byte 0

Bits	Description
07:00	GTC Offset Value Bits [7:0]

786h SET GOF1 - Set GTC Offset Value Byte 1

Bits	Description
07:00	GTC Offset Value Bits [15:8]

787h SET GOF2 - Set GTC Offset Value Byte 2

Bits	Description
07:00	GTC Offset Value Bits [23:16]

788h SET GOF3 - GTC Offset Value Byte 3

Bits	Description
07:00	GTC Offset Value Bits [31:24]

789hF89h SETGET_GDI - SetGet GTC Device Index

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present) This is the pipe based. 000000 i s pipeA and so on and so forth.

3h SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.

Bits	Bits	Description
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

B8h GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	0	*Reserved*
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
0Fh	PARAM_PS	Suppoted Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.

Bits	Reset	Description
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	0	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio
26	0	*Reserved*
25	0	*MST capable (DP1.2)*
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	0	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI

Bits	Reset	Description
06:05	0	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	0	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	0	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	0	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.

Bit	Reset	Description
00	1	**D0 Supported (D0S):** Indicates support for D0.

701hF01h SETGET_CSC - SetGet Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC)**

F02h GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:24	0	Reserved
23:16	100	**Connection List Entry (CLE):** Hardwired to Converter ID 4
15:08	011	**Connection List Entry (CLE)**: Hardwired to Converter ID 3
07:00	010	**Connection List Entry (CLE)**: Hardwired to Converter ID 2

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.

Bits	Reset	Description
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707hF07h SETGET_PWC - SetGet Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT)**

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG)**

F09h GET_PS - Get Pin Sense

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV)**
29	0	**Inactive (INA)**
28:00	28:00	Reserved

71Ch SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	**Default Association (DA)**
03:00	**Sequence (SEQ)**

71Dh SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	**Color (COL)**
03:00	**Miscellaneous (MISC)**

71Eh SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	**Default Device (DD)**
03:00	**Connection Type (CT)**

71Fh SET_CD3 - Set Configuration Default Byte 3

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>07:06</td><td>Port Connectivity (PC): External connectivity of the pin complex.
00 = Connected to jack
01 = No physical connection
10 = Fixed function device (integrated speaker, mic, etc.)
11 = Both a jack and internal connection</td></tr>
<tr><td>05:00</td><td>Location (LOC):
<table>
<tr><td></td><td colspan="10">Bits 3:0</td></tr>
<tr><td>Bits 5:4</td><td>0h: N/A</td><td>1h: Rear</td><td>2h: Front</td><td>3h: Left</td><td>4h: Right</td><td>5h: Top</td><td>6h: Bottom</td><td>7h: Special</td><td>8h: Special</td><td>9h: Special</td><td>Ah-Fh Reserved</td></tr>
<tr><td>00: External</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
<tr><td>01: Internal</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td></tr>
<tr><td>10: Separate Chassis</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11: Other</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
</table></td></tr>
<tr><td></td><td></td></tr>
</table>

F1Ch GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Fh Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734hF34h SETGET_CCM - GetSet Converter Channel Map

Bits	Reset	Description
07:04	0h	**Converter Channel (CC)**
03:00	0h	**Slot (SN)**

740hF40h SETGET PTS Offset Byte0

Bits	Reset	Description
07:00	00h	**PTS B0(PTSB0):** Bits 7:0 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

741hF41h SETGET PTS Offset Byte1

Bits	Reset	Description
07:00	00h	**PTS B1(PTSB1):** Bits 15:8 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

742hF42h SETGET PTS Offset Byte2

Bits	Reset	Description
07:00	00h	**PTS B2(PTSB2):** Bits 23:16 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

743hF43h SETGET PTS Offset Byte3

Bits	Reset	Description
07:00	00h	**PTS B3(PTSB3):** Bits 31:24 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

North Display Engine Registers

This chapter contains the register descriptions for the display portion of a family of graphics devices.

These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

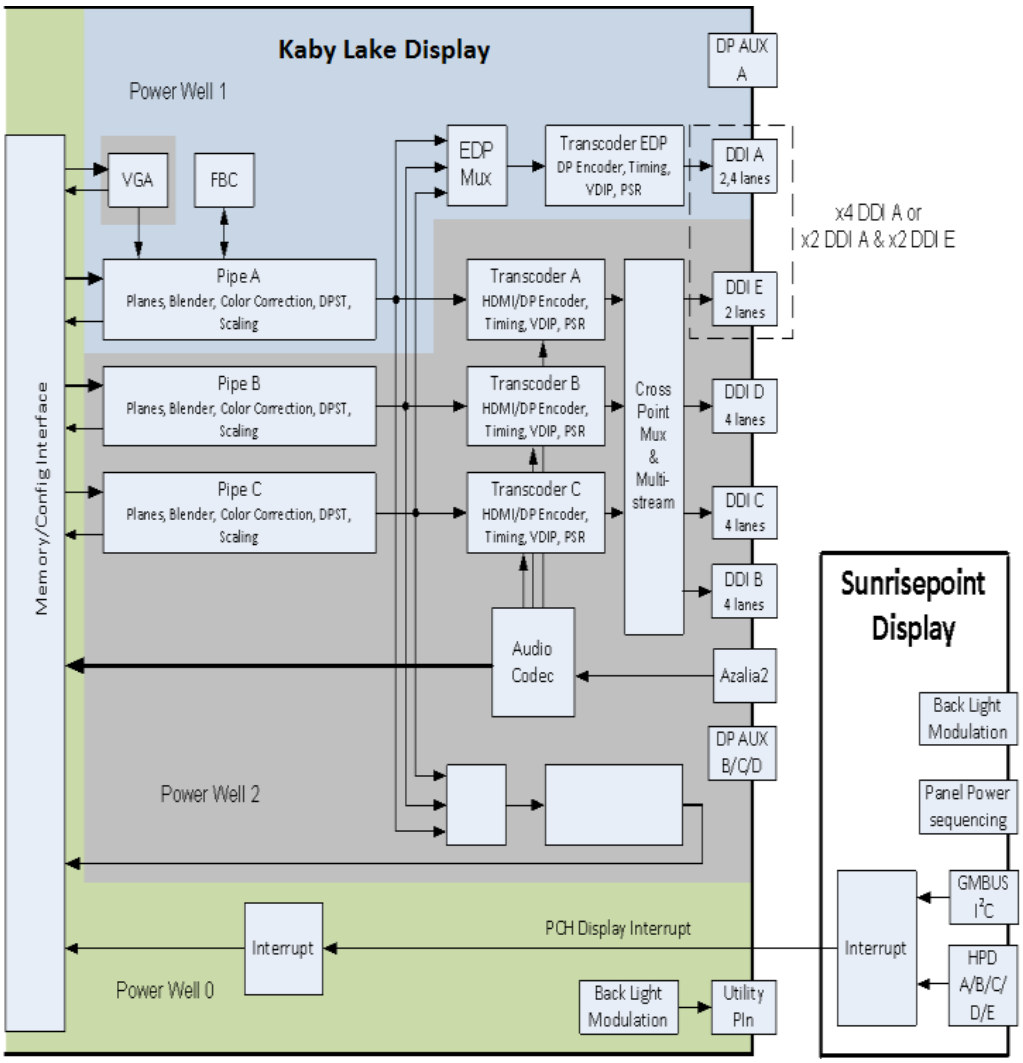
Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

This chapter contains the register descriptions for the display portion of a family of graphics devices.

These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

Display Connections

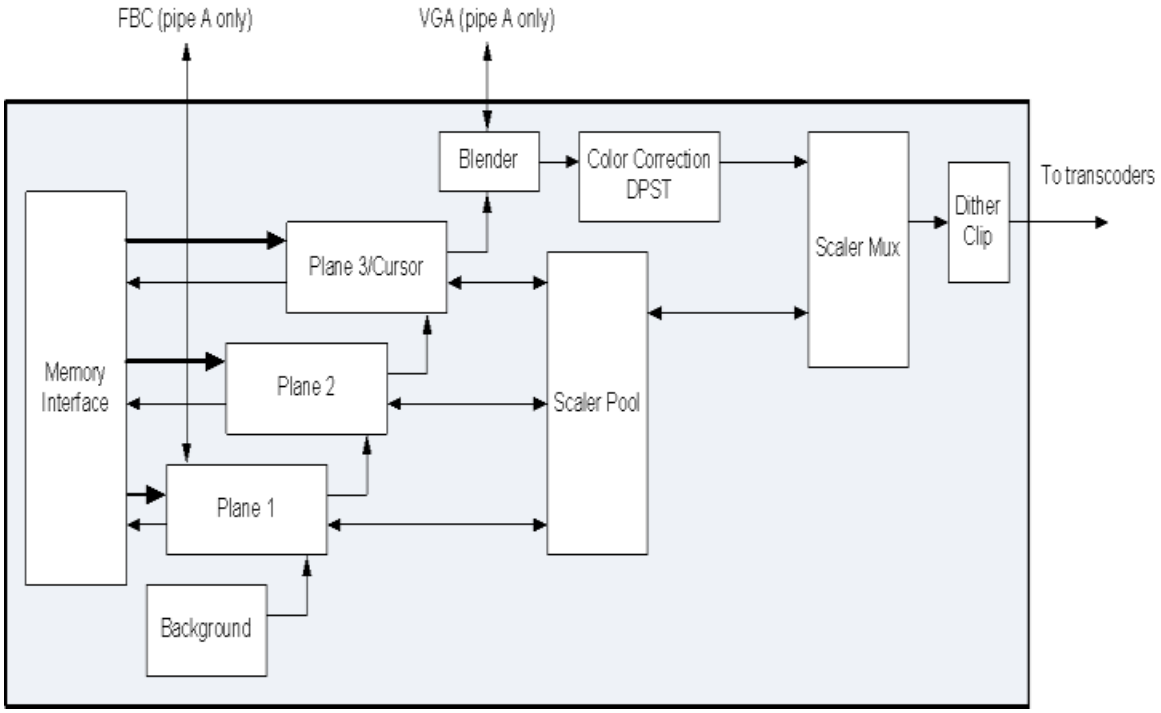


The front end of the display contains the pipes. There are three instances which are referred to as Pipe A, Pipe B, and Pipe C.

The pipes connect to the transcoders. There are five instances which are referred to as Transcoder A, Transcoder B, Transcoder C, Transcoder EDP, and Transcoder WD0.

The transcoders connect to the DDIs. There are five instances which are referred to as DDI A, DDI B, DDI C, DDI D, and DDI E.

Display Pipes



The display pipes contain the planes, blending, color correction, DPST, scaling, dithering, and clipping.

Each display pipe has three planes and a cursor. Each plane can be used as a sprite, primary, or overlay.

The background color that is seen under the bottom most plane is programmable.

The plane blending follows a fixed Z-order. Plane 1 is the bottom most plane and higher numbered planes stack on top of it.

The cursor and top most plane are mutually exclusive and cannot be both enabled at the same time.

Pipe A and Pipe B each have two pipe scalers. Pipe C has one pipe scaler.

Each pipe scaler can be assigned to scale an individual plane or scale the blended output.

Display Transcoders

The display transcoders contain the timing generators, port encoders, Audio/Video mixers, Video Data Island Packet mixers, and Panel Self Refresh controllers.

Transcoder EDP does not support HDMI, DVI, or Audio.

Transcoder WD0 only supports display capture and write back to memory.

Audio

The Azalia interface provides data to the audio codec.

The audio codec connects to the Audio/Video mixers in the transcoders.

DDIs

The DDIs contain the DisplayPort transport control and other port logic to interface to the DDI physical pins.

DDI A, DDI B, DDI C, and DDI D support lane reversal where the internal lane to package lane mapping is swapped.

DDI E does not support lane reversal.

DDI A and DDI E do not support DisplayPort multistream.

DDI A and DDI E share 2 lanes. DDI A is capable of supporting up to 4 lanes when DDI E is not connected, but only 2 lanes when DDI E is connected. DDI E is capable of supporting up to 2 lanes when connected. Dynamic switching between the two configurations is not supported.

DisplayPort A and E Lane Mapping:

Package Pin	Non-Reversed when DDI A is x4 capable	Reversed when DDI A is x4 capable	Non-Reversed when DDI A is x2 capable	Reversed when DDI A is x2 capable
DDIA/DDIE 3	Port A Main Link Lane 3	Port A Main Link Lane 0	Port E Main Link Lane 1	Port E Main Link Lane 1
DDIA/DDIE 2	Port A Main Link Lane 2	Port A Main Link Lane 1	Port E Main Link Lane 0	Port E Main Link Lane 0
DDIA/DDIE 1	Port A Main Link Lane 1	Port A Main Link Lane 2	Port A Main Link Lane 1	Port A Main Link Lane 0
DDIA/DDIE 0	Port A Main Link Lane 0	Port A Main Link Lane 3	Port A Main Link Lane 0	Port A Main Link Lane 1

DisplayPort B, C, and D Lane Mapping:

Package Pin	Non-Reversed	Reversed
DDI 3	Main Link Lane 3	Main Link Lane 0
DDI 2	Main Link Lane 2	Main Link Lane 1
DDI 1	Main Link Lane 1	Main Link Lane 2
DDI 0	Main Link Lane 0	Main Link Lane 3

HDMI/DVI TMDS Lane Mapping:

Package Pin	Non-Reversed	Reversed
DDI 3	TMDS Clock	TMDS Data2
DDI 2	TMDS Data0	TMDS Data1
DDI 1	TMDS Data1	TMDS Data0
DDI 0	TMDS Data2	TMDS Clock

DDI Equivalent Names:

DDI Name	Equivalent Names
DDI A	Port 0, DDI-A, DDIA, Port A, DDI0, eDP
DDI B	Port 1, DDI-B, DDIB, Port B, DDI1
DDI C	Port 2, DDI-C, DDIC, Port C, DDI2
DDI D	Port 3, DDI-D, DDID, Port D, DDI3
DDI E	Port 4, DDI-E, DDIE, Port E, DDI4

Pipe to Transcoder to DDI Mappings

Twin modes are not supported.

Any pipe can drive any single DDI.

With DisplayPort multistream it is possible to have multiple pipes driving a single DDI. DDI B, DDI C, and DDI D support multistream. DDI A and DDI E do not support multistream.

Pipe A can connect to either Transcoder A, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Pipe B can connect to either Transcoder B, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Pipe C can connect to either Transcoder C, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Transcoder A is tied to Pipe A.

Transcoder B is tied to Pipe B.

Transcoder C is tied to Pipe C.

Transcoder EDP can connect to Pipe A, Pipe B, or Pipe C, but only one at a time.

Transcoder WD0 can connect to Pipe A, Pipe B, or Pipe C, but only one at a time.

Transcoder A can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder B can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder C can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder EDP can connect only to DDI A.

Transcoder WD0 does not connect to any DDI. Transcoder WD0 output only goes to memory write back.

DDI A can connect only to Transcoder EDP. DDI A does not support DisplayPort multistream.

DDI B can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI C can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI D can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI E can connect to Transcoder A, Transcoder B, or Transcoder C, individually.

Terminology

Term	Description
DP	DisplayPort
SST, DP SST	DisplayPort Single Stream Transport
MST, DP MST	DisplayPort Multi Stream Transport

Register Access Field	Description	Implementation
R/W (Read/Write)	The value written into this register will control hardware and is the same value that will be read.	Write data is stored. Read is from the stored data. Stored value is used to control hardware.
Reserved	Unused register bit. Don't assume a value for these bits. Writes have no effect.	Write data is ignored. Read is zero.
MBZ (Must Be Zero)	Always write a zero to this register.	May be implemented as Reserved or as R/W.
PBC (Preserve Bit Contents)	Software must write the original value back to this bit. This allows new features to be added using these bits.	May be implemented as Reserved or as R/W.
Read Only	The read value is determined by hardware. Writes to this bit have no effect.	Write data is ignored. Read is from a status signal or some other internal source.
Write Only	The value written into this register will control hardware. Reads return zero.	Write data is stored. Read is zero. Stored value is used to control hardware.
R/W Clear (Read/Write Clear)	Sticky status bit. Hardware will set the bit, software can clear it with a write of 1b.	Internal hardware events set a sticky bit. Read is from the sticky bit. A write of 1b clears the sticky bit.
R/W Set (Read/Write Set)	Sticky status bit. Software can set the bit with a write of 1b. Hardware will clear the bit.	A write of 1b sets a sticky bit. Internal hardware events clear a sticky bit. Read is from the sticky bit.

Register Access Field	Description	Implementation
Double Buffered	Write when desired and the written value will take effect at the time of the update point specified in the 'Double Buffer Update Point' parameter. Reads will return the written value, which is not necessarily the value being currently used to control hardware. Some have a specific arming sequence where a write to another register, specified in the 'Double Buffer Armed By' parameter, is required before the update can take place. Once the armed by register is written to, the written values of all registers controlled by that arming will take effect at the time of the double buffer update point. This is used to ensure atomic updates of several registers. Note: Once armed, by write to the armed by register, the registers controlled by this arming should not be changed until the double buffer update point is reached. If changed, this will disarm the sequence and will require another write to the armed by register to get it to the armed status again.	Two stages of registers used. Write data is stored into first stage. Read is from the first stage stored data. First stage stored value is transferred to second stage storage at the double buffer update point. Second stage stored value is used to control hardware. Arm/disarm logic may be used for some registers to control the double buffer update point.
Write/Read Status	The value written into this register will control hardware. The read value is determined by hardware.	Write data is stored. Stored value is used to control hardware. Read is from a status signal or some other internal source.

Mode Set

A mode set sequence is the programming sequence that must be followed when enabling or disabling output to a display. There are several different mode set sequences documented in the following sections. The sequence to use depends on which type of port is being enabled or disabled.

Sequences to Initialize Display

These sequences are used to initialize the display engine before any display engine functions can be enabled.

Most display engine functions will not operate while display is not initialized. Only basic PCI, I/O, and MMIO register read/write operations are supported when display is not initialized.

Initialize Sequence

1. Enable PCH Reset Handshake
    a. Set NDE_RSTWRN_OPT RST PCH Handshake En to 1b.
2. Enable Power Well 1 (PG1) and Misc IO Power
    a. Poll for FUSE_STATUS Fuse PG0 Distribution Status = 1b.
        - Timeout and fail after 5 us.
    b. Set PWR_WELL_CTL Power Well 1 Request and Misc IO Power Request to 1b.
    c. Poll for PWR_WELL_CTL Power Well 1 State and Misc IO Power State = 1b.
        - Timeout and fail after 10 us.
    d. Poll for FUSE_STATUS Fuse PG1 Distribution Status = 1b.
        - Timeout and fail after 5 us.
3. Enable CDCLK PLL
    a. Set CDCLK_CTL CD Frequency Select to the minimum.
    b. Configure DPLL0 link rate to the rate required for the eDP panel, or 810 MHz (DP 1.62 GHz) if there is no eDP panel.
    c. Set LCPLL1_CTL PLL enable to 1b.
    d. Poll for LCPLL1_CTL PLL lock = 1b.
        - Timeout and fail after 5 ms.
4. Follow Sequences for Changing CD Clock Frequency
5. Enable DBUF
    a. Set DBUF_CTL DBUF Power Request to 1b.
    b. Poll for DBUF_CTL DBUF Power State = 1b.
        - Timeout and fail after 10 us.

Un-initialize Sequence

1. Disable all display engine functions using the full mode set disable sequence on all pipes, transcoders, ports, planes, and power well 2 (PG2).
2. Disable DBUF
    a. Clear DBUF_CTL DBUF Power Request to 0b.
    b. Poll for DBUF_CTL DBUF Power State = 0b.
        - Timeout and fail after 10 us.
3. Disable CDCLK PLL
    a. Clear LCPLL1_CTL PLL enable to 0b.
    b. Poll for LCPLL1_CTL PLL lock = 0b.
        - Timeout and fail after 1 ms.
4. Disable Power Well 1 (PG1) and Misc IO Power
    a. Clear PWR_WELL_CTL Power Well 1 Request and Misc IO Power Request to 0b.
    b. Wait for 10us. Do not poll for the power well to disable. Other clients may be keeping it enabled.

Sequences for DisplayPort

This topic describes how to enable and disable DisplayPort.

Enable Sequence

Display must already be initialized

DDIA Lane Capability Control must be configured prior to enabling any ports or port clocks

1. **Enable Power Wells**

If not PipeA+DDIA - Enable Power Well 2 a. Enable PWR_WELL_CTL Power Well 2 Request b. Wait for PWR_WELL_CTL Power Well 2 State = Enabled, timeout after 20 us c. Wait for FUSE_STATUS FUSE PG2 Distribution Status = Done, timeout after 1 us

2. **If panel power sequencing is required - Enable Panel Power**
    a. Enable panel power sequencing
    b. Wait for panel power sequencing to reach the enabled state
3. **Enable Port PLL**
    a. If PLL is not already enabled, follow port clock programming sequence from Clocks section
    b. If PLL to port mapping is flexible, configure PLL to port mapping to direct the PLL output to the DDI

4. **If IO power is controlled through PWR_WELL_CTL - Enable IO Power**
    a. Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
    b. Wait for PWR_WELL_CTL DDI IO Power Request = Enabled, timeout after 20 us
5. **Enable and Train DisplayPort**
    a. Configure and enable DP_TP_CTL with link training pattern 1 selected
    b. Configure voltage swing and related IO settings. Refer to DDI Buffer section.
    c. Configure and enable DDI_BUF_CTL
    d. Gen9: Wait >518 us for buffers to enable before starting training or allow for longer time in TP1 before software timeout.
    e. Follow DisplayPort specification training sequence (see notes for failure handling)
    f. If DisplayPort multi-stream - Set DP_TP_CTL link training to Idle Pattern, wait for 5 idle patterns (DP_TP_STATUS Min_Idles_Sent) (timeout after 800 us)
    g. Set DP_TP_CTL link training to Normal.
6. **Enable Planes, Pipe, and Transcoder (repeat to add multiple pipes on a single port for multi-streaming)**
    a. If DisplayPort multi-stream - use AUX to program receiver VC Payload ID table to add stream
    b. Configure Transcoder Clock Select to direct the Port clock to the Transcoder
    c. Configure and enable planes (VGA or hires). This can be done later if desired.
    d. If VGA - Clear VGA I/O register SR01 bit 5
    e. Enable panel fitter if needed (must be enabled for VGA)
    f. Configure transcoder timings, M/N/TU/VC payload size, and other pipe and transcoder settings
    g. Configure and enable TRANS_DDI_FUNC_CTL
    h. If DisplayPort multistream - Enable pipe VC payload allocation in TRANS_DDI_FUNC_CTL
    i. If DisplayPort multistream - Wait for ACT sent status in DP_TP_STATUS and receiver DPCD (timeout after >410us)
    j. Configure and enable TRANS_CONF
    k. If panel power sequencing is required - Enable panel backlight

SRD and/or Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Notes

Changing voltage swing during link training:

- Change the swing setting following the DDI Buffer section. The port does not need to be disabled.

Changing port width (lane count) or frequency during link training:

1. Follow Disable Sequence for DisplayPort to Disable Port

2. If PLL frequency needs to change, Follow Disable Sequence for DisplayPort to Disable PLL, then follow Enable Sequence for DisplayPort to Enable PLL, using the new frequency settings
3. Follow Enable Sequence for DisplayPort to Enable and Train DisplayPort, using the new port width settings

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Enabling DisplayPort Sync Mode

See TRANS_DDI_FUNC_CTL Port Sync Mode Enable for restrictions.

1. Follow the enable sequence for the DisplayPort slave, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
    - Set TRANS_DDI_FUNC_CTL Port Sync Mode Master Select and Port Sync Mode Enable when configuring and enabling TRANS_DDI_FUNC_CTL.
2. Wait 200 uS.
3. Follow the enable sequence for the DisplayPort master, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
4. Set DisplayPort slave DP_TP_CTL link training to Normal.
5. Set DisplayPort master DP_TP_CTL link training to Normal.

Software may need to use DOUBLE_BUFFER_CTL to ensure updates to plane and pipe registers will take place in the same frame.

For example: If pipe A and pipe B are synchronized together and software needs the surface addresses for two planes to update at the same time, software should use DOUBLE_BUFFER_CTL when writing the surface address registers for both planes, otherwise there is a possibility that the updates could be split across a vertical blank such that one plane would update on the current vertical blank and the other plane would update on the next vertical blank.

Disable Sequence

SRD and Audio must be disabled first. Follow the audio disable sequence in the audio registers section.

1. **If panel power sequencing is required - Disable panel backlight**
2. **Disable Planes, Pipe, and Transcoder (repeat to remove multiple pipes from a single port for multi-streaming)**
    a. If VGA
        i. Set VGA I/O register SR01 bit 5 for screen off
        ii. Wait for 100 us
    b. Disable planes (VGA or hires)
    c. Disable TRANS_CONF

d. Wait for off status in TRANS_CONF, timeout after two frame times
e. If DisplayPort multistream - use AUX to program receiver VC Payload ID table to delete stream
f. If done with this VC payload
    i. Disable VC payload allocation in TRANS_DDI_FUNC_CTL
    ii. Wait for ACT sent status in DP_TP_STATUS and receiver DPCD
g. Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
h. Disable panel fitter
i. Configure Transcoder Clock Select to direct no clock to the transcoder

3. **Disable Port (all pipes and VC payloads on this port must already be disabled)**
    a. Disable DDI_BUF_CTL
    b. Disable DP_TP_CTL (do not set port to idle when disabling)
    c. Gen9: Wait 8 us or poll on DDI_BUF_CTL Idle Status for buffers to return to idle
4. **If panel power sequencing is required - Disable Panel Power**
    a. Disable panel power sequencing
5. **If IO power is controlled through PWR_WELL_CTL - Disable IO Power**
    a. Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used
6. **Disable Port PLL**
    a. If PLL to port mapping is flexible, configure PLL to port mapping to direct no clock to the DDI
    b. If this PLL is no longer needed, follow PLL disable sequence from Clocks section
7. **Disable Power Wells**
    a. If no required resource is in the power well - Disable PWR_WELL_CTL Power Well 2 Request

Disabling DisplayPort Sync Mode

1. Follow the disable sequence for the DisplayPort slave.
2. Follow the disable sequence for the DisplayPort master.

Sequences for HDMI and DVI

This topic describes how to enable and disable HDMI and DVI.

Enable Sequence

Display must already be initialized

1. **Enable Power Wells**
   a. Enable PWR_WELL_CTL Power Well 2 Request
   b. Wait for PWR_WELL_CTL Power Well 2 State = Enabled, timeout after 20 us
   c. Wait for FUSE_STATUS FUSE PG2 Distribution Status = Done, timeout after 1 us
2. **Enable Port PLL**
   a. If PLL is not already enabled, follow port clock programming sequence from Clocks section
   b. If PLL to port mapping is flexible, configure PLL to port mapping to direct the PLL output to the DDI
3. **If IO power is controlled through PWR_WELL_CTL - Enable IO Power**
   a. Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
   b. Wait for PWR_WELL_CTL DDI IO Power Request = Enabled, timeout after 20 us
4. **Enable Planes, Pipe, and Transcoder**
   a. Configure Transcoder Clock Select to direct the Port clock to the Transcoder
   b. Configure and enable planes (VGA or hires). This can be done later if desired.
   c. If VGA - Clear VGA I/O register SR01 bit 5
   d. Enable panel fitter if needed (must be enabled for VGA)
   e. Configure transcoder timings and other pipe and transcoder settings
   f. Configure and enable TRANS_DDI_FUNC_CTL
   g. Configure and enable TRANS_CONF
5. **Enable Port**
   a. Configure voltage swing and related IO settings. Refer to the DDI Buffer section.
   b. Configure and enable DDI_BUF_CTL

Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Notes

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Disable Sequence

Audio must be disabled first. Follow the audio disable sequence in the audio registers section.

1. **Disable Planes, Pipe, and Transcoder**
    a. If VGA
        i. Set VGA I/O register SR01 bit 5 for screen off
        ii. Wait for 100 us
    b. Disable planes (VGA or hires)
    c. Disable TRANS_CONF
    d. Wait for off status in TRANS_CONF, timeout after two frame times
    e. Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
    f. Disable panel fitter
    g. Configure Transcoder Clock Select to direct no clock to the transcoder
2. **Disable Port**
    a. Disable DDI_BUF_CTL
3. **If IO power is controlled through PWR_WELL_CTL - Disable IO Power**
    a. Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used used
4. **Disable Port PLL**
    a. If PLL to port mapping is flexible, configure PLL to port mapping to direct no clock to the DDI
    b. If this PLL is no longer needed, disable it
5. **Disable Power Wells**

a. If no required resource is in the power well - Disable PWR_WELL_CTL Power Well 2 Request

Sequences for Display C5 and C6

Display C5 (DC5) is a power saving state where hardware dynamically disables power well 1 and the CDCLK PLL and saves the associated registers.

DC5 can be entered when software allows it, power well 2 is disabled, and hardware detects that all pipes are disabled or pipe A is enabled with PSR active.

Display C6 (DC6) is a deeper power saving state where hardware dynamically disables power well 0 and saves the associated registers.

DC6 can be entered when software allows it, the conditions for DC5 are met, and the PCU allows DC6.

DC6 cannot be used if the backlight is being driven from the display utility pin.

Core CPUs support DC5 and DC6.

The context save and restore program is reset on cold boot, warm reset, PCI function level reset, and hibernate/suspend.

Sequence to Allow DC5 or DC6

1. Load the correct stepping specific Display Context Save and Restore (CSR) program from the binary package.
    a. Read the package header and extract the correct individual firmware. Binary package format details can be found in sections below.
    b. Skip the header section at the start of the program binary.
    c. Copy the payload into **Display CSR Program Storage** .
    d. Perform the MMIO writes specified in the header section.
2. Configure display engine to have power well 2 disabled, following the appropriate mode set disable sequences for any ports using power well 2. This can be done earlier if desired.
3. Set display register 0x45520 bit 1 to 1b. It does not need to be cleared at any time.
4. Set DC_STATE_EN Dynamic DC State Enable = "Enable up to DC5" for DC5 or "Enable up to DC6" for DC6.

Programming Note	
Context:	Display C5 and C6
Do not switch between "Enable up to DC5" and "Enable up to DC6" without moving to "Disable" and reloading the CSR program in between. Disable DC5/DC6 during mode set and re-enable after the mode set programming is completed. MMIO accesses have more latency when DC5/DC6 is enabled. For optimal performance, disable DC5/DC6 when programming a set of registers and re-enable them after the programming is completed.	

Sequence to Disallow DC5 and DC6

1. Set DC_STATE_EN Dynamic DC State Enable = "Disable".

DMC Firmware Package

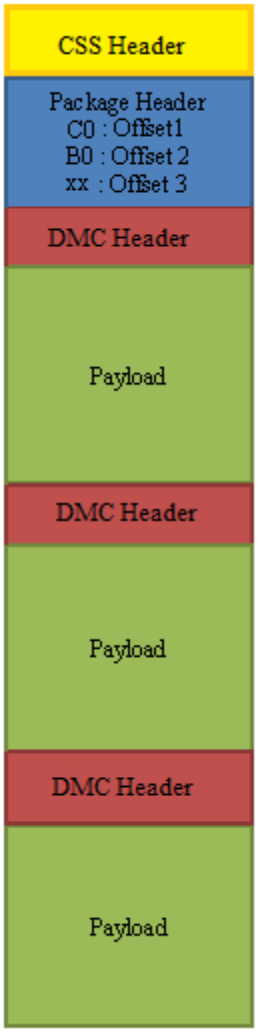
Display Micro-Controller firmware package includes all the firmwares that are required for different steppings of the product. The stepping dependent firmwares are all packaged and released as a single binary package. The package contains the CSS header, followed by the package header and the actual DMC firmwares.

Packaged firmware uses the following naming convention - <project>_dmc_ver<major>_<minor>.bin. The major version will get incremented whenever there is a change in the header layout and would require an update to the driver firmware loading module.

Major version 1

CSS Header	1.0
Package Header	1
DMC Header	1

Package Layout



CSS Header

```
typedef struct _CssHeader {
      uint32_t     moduleType;                    // 0x09 for DMC
      uint32_t     headerLen;                     // CSS header length in dwords
      uint32_t     headerVer;                     // 0x10000
      uint32_t     moduleID;                      // Not used
      uint32_t     moduleVendor;                  // Not used
      uint32_t     date;                          // YYYYMMDD(YYYY « 16 + MM « 8
+ DD)
      uint32_t     size;                          // Total dmc fw binary size in
dwords - (CSS_Headerlen + PackageHeaderLen + dmc FWsLen)/4
      uint32_t     keySize;                       // Not used
      uint32_t     modulusSize;                   // Not used
      uint32_t     exponentSize;                  // Not used
      uint32_t     reserved1[12];                 // Not used
      uint32_t     version;                       // Major Minor
      uint32_t     reserved2[8];                  // Not used
      uint32_t     uKernelHeaderInfo;             // Not used
} CssHeader;
```

Package Header

Package header contains the firmware/stepping mapping table and the corresponding firmware offsets to the individual binaries, within the package. Mapping table will list the exceptions first, followed by the default entries. An Offset value of "0xFFFFFFFF" in the mapping table indicates that there is no firmware available/supported for that stepping. The offsets to the individual binary are DWord aligned. The first individual binary starts at an offset value of "0x00000000" after the CSS Header and the Package Header.

Stepping/Version mapping example

Stepping	FW Version
A1	1.1
B*	1.6
**	2.3

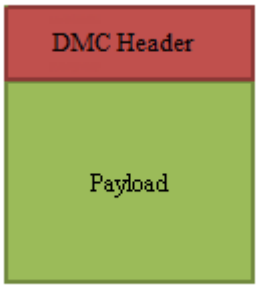
```
typedef struct _PackageHeader {
      uint8_t      headerLen;                     // DMC package header length
in dwords
      uint8_t      headerVer;                     // 0x01
```

```
      uint8_t      reserved[10];                    // Reserved
      uint32_t     numEntries;                      // Number of valid entries in
the FWInfo array below
      struct _FWInfo_ {
            uint16_t     reserved1;                 // Reserved
            char         stepping;                  // Stepping (A, B, C, ..., *).
* is a wildcard
            char         substepping;               // Sub-stepping (0, 1, ...,
*). * is a wildcard
            uint32_t     offset;                    // FW offset within the
package
            uint32_t     reserved2;                 // Reserved
      } FWInfo[20];
} PackageHeader;
```

DMC firmware binary

Each individual DMC firmware binary has a header followed by a payload whose size is specified in the header section. Along with the version, length, firmware size etc. the header section also specifies a list of MMIO addresses and data. These MMIO write cycles, in the 0x80000 - 0x8FFFF address range, should be executed as part of the initial CSR program setup.

DMC firmware Layout




```
for i = 1 to <mmioCount>
      Perform MMIO write to address <mmioaddr[i]> with data <mmiodata[i]>
typedef struct _DMCHeader {
     uint32_t    reserved;              // 0x40403E3E
     uint8_t     headerLen;             // DMC binary header length in bytes
     uint8_t     headerVer;             // 0x01
     uint16_t    dmccVer;               // Reserved
     uint32_t    project;               // Major, Minor
```

```
    uint32_t    fwSize;                 // Firmware program size (excluding header)
    in dwords
    uint32_t    fwVersion;              // Major Minor version
    uint32_t    mmioCount;              // Number of valid MMIO cycles present in
    the MMIO address and data arrays below.
    uint32_t    mmioaddr[8];            // MMIO address
    uint32_t    mmiodata[8];            // MMIO data
    unint8_t    dfile[32];              // Reserved
    uint32_t    reserved1[2];           // Reserved
} DMCHeader;
```

Display Resolution Support

A display resolution is only supported if it meets all the restrictions below for Maximum Pipe Pixel Rate, Maximum Port Link Rate, Maximum Memory Read Bandwidth, and Maximum Watermark.

Maximum Pipe Pixel Rate

The display resolution must fit within the maximum pixel rate output from the pipe. Make sure that the display pipe is able to feed pixels at a rate required to support the desired resolution.

For each enabled plane on the pipe {

If plane scaling enabled {

Horizontal down scale amount = Maximum[1, plane horizontal size / scaler horizontal window size]

Vertical down scale amount = Maximum[1, plane vertical size / scaler vertical window size]

Plane down scale amount = Horizontal down scale amount * Vertical down scale amount

Plane Ratio = 1 / Plane down scale amount

}

Else {

Plane Ratio = 1

}

If plane source pixel format is 64 bits per pixel {

Plane Ratio = Plane Ratio * 8/9

}

}

Pipe Ratio = Minimum Plane Ratio of all enabled planes on the pipe

If pipe scaling is enabled {

Horizontal down scale amount = Maximum[1, pipe horizontal source size / scaler horizontal window size]

Vertical down scale amount = Maximum[1, pipe vertical source size / scaler vertical window size]

Note: The progressive fetch - interlace display mode is equivalent to a 2.0 vertical down scale

Pipe down scale amount = Horizontal down scale amount * Vertical down scale amount

Pipe Ratio = Pipe Ratio / Pipe down scale amount

}

Pipe maximum pixel rate = CDCLK frequency * Pipe Ratio // See the display clocks section for the supported CDCLK frequencies.

Maximum Port Link Rate

The display resolution must fit within the maximum link rate for each port type.

Port Type	Maximum Link Bit Rate
eDP/DP	HBR2 5.4 GHz
HDMI	3 GHz
DVI	1.65 GHz
MIPI DSI	1.066 GHz

Maximum Memory Read Bandwidth

The display resolution must not exceed the available system memory bandwidth, considering factors like thermal throttling and bandwidth available for other memory clients.

For each pipe {

For each plane enabled on the pipe { // cursor can be ignored

Plane bandwidth MB/s = pixel rate MHz * source pixel format in bytes * plane down scale amount * pipe down scale amount

Total display bandwidth MB/s = Total display bandwidth + Plane bandwidth

}

}

Raw system memory bandwidth = (# of memory channels * memory frequency * 8 bytes)

If Total display bandwidth > system memory bandwidth available for display {Bandwidth is exceeded}

Programming Note	
Context:	Display Resolution Support
Do not use more than 60% of raw system memory bandwidth for display.	

Maximum Watermark

The display resolution must not exceed the level 0 maximum watermark value. See the volume on Watermark Programming.

Display Resolution Capabilities

These are examples of common resolutions that meet all the resolution restrictions for up to 3 simultaneous displays, 4 primary or sprite planes with 32bpp pixel format, and 1 cursor, with no panel fitter down scaling.

Attribute	CD 675 MHz	CD 540 MHz	CD 450 MHz	CD 337.5 MHz
eDP/DP x4 single stream	3840x2560 60Hz 24Bpp[4] 4096x2304 60Hz 24Bpp[4]	3840x2160 60Hz 30Bpp[4]	3200x2000 60Hz 30Bpp[3]	2880x1620 60Hz 30Bpp[2] 4096x2160 30Hz 30Bpp[2]
HDMI	Same as entry to the right	Same as entry to the right	Same as entry to the right	4Kx2K 24-30Hz 24Bpp[2]
DVI	Same as entry to the right	Same as entry to the right	Same as entry to the right	1920x1200 60Hz 24Bpp[1]

Each entry is showing the highest common resolutions at that CD clock frequency step. Lower resolutions are also supported. Higher, less common, resolutions can also work, but need to be calculated individually.

eDP, DP, DVI, and MIPI DSI are calculated using CVT 1.2 RB1 blanking and pixel rate.

HDMI is calculated using HDMI specification blanking and pixel rate.

Bpp is referring to the port output bits per pixel.

[1]Requires at least single channel DDR3 1333 for 3 simultaneous displays

[2]Requires at least single channel DDR3 1600 for 3 simultaneous displays

[3]Requires at least dual channel DDR3 1333 for 3 simultaneous displays

[4]Requires at least dual channel DDR3 1600 for 3 simultaneous displays

Examples

Example pipe pixel rate:

Plane 1 enabled at 32bpp, plane 2 enabled at 16bpp, pipe scaling enabled and down scale amount 1.12, and CDCLK 450 MHz:

Plane 1 ratio = 1

Plane 2 ratio = 1

Pipe ratio = Minimum[1, 1] = 1

Pipe ratio = 1/1.12 = 0.89

Pipe maximum pixel rate = 450 MHz * 0.89 = 400.5 MHz

Example pipe pixel rate:

Plane 1 enabled at 64bpp and plane down scale amount 1.25, plane 2 enabled at 32bpp, no panel fitting enabled, and CDCLK 540 MHz:

Plane 1 ratio = 1/1.25 * 8/9 = 0.71

Plane 2 ratio = 1

Pipe ratio = Minimum[1, 0.71] = 0.71

Pipe maximum pixel rate = 540 MHz * 0.71 = 383.4 MHz

Example memory bandwidth:

System memory bandwidth available for display = 4000 MB/s

Pipe A - Plane 1 enabled at 32bpp, plane 2 enabled at 16bpp, scaling disabled, pixel rate 148.5 MHz

Pipe B - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 148.5 MHz

Pipe C - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 148.5 MHz

Pipe A - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Pipe A - Plane 2 bandwidth = 148.5 * 2 bytes = 297 MB/s

Pipe B - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Pipe C - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Total display bandwidth = 594 + 297 + 594 + 594 = 2079 MB/s

System memory bandwidth available for display not exceeded

Example memory bandwidth:

System memory bandwidth available for display = 4000 MB/s

Pipe A - Plane 1 enabled at 32bpp, plane 2 plane enabled at 32bpp, pipe scaling enabled and down scale amount 1.12, pixel rate 414.5 MHz

Pipe B - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 414.5 MHz

Pipe C - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 414.5 MHz

Pipe A - Plane 1 bandwidth = 414.5 * 4 bytes * 1.12 = 1863 MB/s

Pipe A - Plane 2 bandwidth = 414.5 * 4 bytes * 1.12 = 1863 MB/s

Pipe B - Plane 1 bandwidth = 414.5 * 4 bytes = 1658 MB/s

Pipe C - Plane 1 bandwidth = 414.5 * 4 bytes = 1658 MB/s

Total display bandwidth = 1863 + 1863 + 1658 + 1658 = 7042 MB/s

System memory bandwidth available for display **exceeded**

Clocks

CDCLK_CTL

LCPLL1_CTL

LCPLL2_CTL

WRPLL_CTL

PORT_CLK_SEL

TRANS_CLK_SEL

DPLL_CTRL1

DPLL_CTRL2

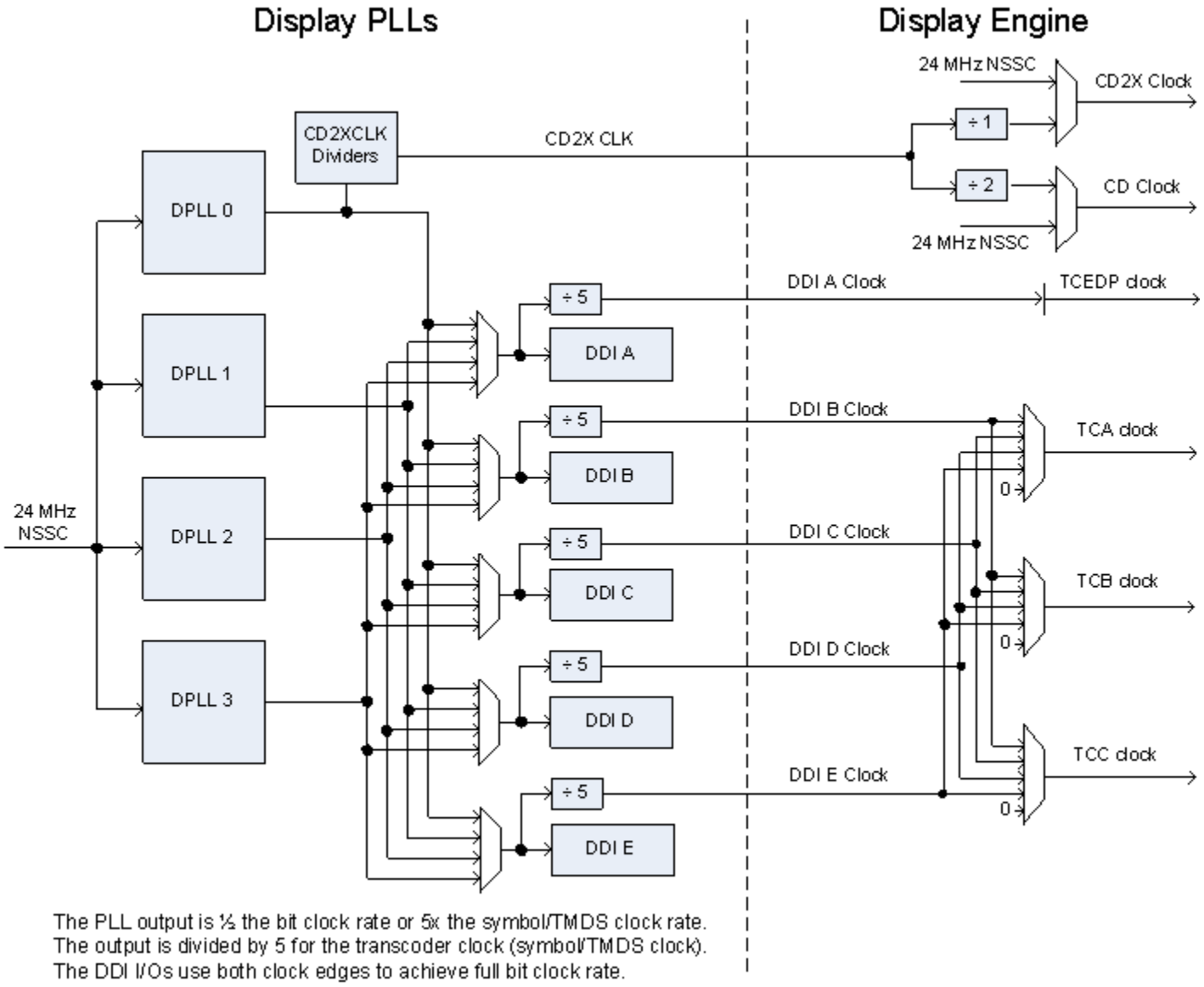
DPLL_STATUS

DPLL_CFGCR1

DPLL_CFGCR2

TIMESTAMP_CTR

Overview of Supported Display Clock Paths



The display engine clocking structure has multiple PLLs and clocks. The flow is from reference to PLL to DDI (port) clock to transcoder clock.

Display Engine Clock Reference

There is one display engine clock reference.

	Non-SSC (NSSC) Reference
Usage	Non-spread spectrum reference for DPLL and alternate source for CD and CD2X clocks
Frequency	24 MHz
Default After Reset	Enabled
Programming	Not programmable by display software.

Display Engine PLLs

There are four display engine PLLs (DPLLs).

The PLL output frequencies are 5x the symbol/TMDS rate, which is 1/2 the bit rate.

The PLL output is divided by 5 to become the symbol/TMDS clock frequency used in the display engine.

Both edges of the PLL output are used in the DDI I/Os to double the frequency to bit clock rate.

A single PLL output may be used by multiple DDI ports simultaneously if those DDI ports all require the same frequency and spread characteristics.

	DPLL0 (LCPLL 1)	DPLL1 (LCPLL 2), DPLL2 (WRPLL 1), DPLL3 (WRPLL 2)
Usage	Sources for DDI clocks and CD clock	Source for DDI clocks
Input	Non-SSC reference	Non-SSC reference
Frequency	Programmable selection in the range 0.1 to 2.7 GHz (0.2 to 5.4 GHz bit rate). CD clock usage does not allow spread spectrum to be used.	Programmable selection in the range 0.1 to 2.7 GHz (0.2 to 5.4 GHz bit rate). Spread spectrum can be added if needed.
Default After Reset	Disabled	Disabled
Programming	DPLL enable and frequency must be programmed by software when enabling and disabling a display. Frequency programming is done through the DPLL_CTRL1 register. Enable is programmed through the LCPLL1_CTL register. See the section on Port Clock Programming.	DPLL enable, SSC enable, and frequency must be programmed by software when enabling and disabling a display. Frequency and SSC programming is done through the DPLL_CTRL1 and DPLL*_CFGCR* registers. Enables are programmed through the LCPLL2_CTL and WRPLL_CTL registers. See the section on Port Clock Programming.

Recommended PLL Selection

Avoid changing frequencies on DPLL 0 since that requires DPLL 0 to be disabled, which impacts CDCLK and requires display features to be disabled.

Use DPLL 2 as a last choice because that PLL can be taken by HDPORT in some cases. See register HDPORT_STATE.

- For eDP Non-SSC, use DPLL 0 with frequency set to match the eDP panel requirement.
- For eDP SSC or external displays, use DPLLs 1, 3, 2 in that order, depending on the availability.
- If DPLL 0 is not used for eDP, run DPLL 0 in DisplayPort mode with link rate 810 MHz (DP 1.62 GHz). This will select VCO 8100 for generating CDCLK.

DDI Clocks

There is one DDI clock tied to each DDI port.

A single DDI clock output may be used by multiple transcoders simultaneously for DisplayPort Multi-streaming.

	DDI clocks A, B, C, D, and E
Usage	DDI ports I/O bit clock and symbol/TMDS clock, source for transcoder clocks
Input	Programmable selection between the DPLLs
Frequency	DPLL frequency
Default After Reset	Disabled
Programming	Mapping of DPLL to DDI must be programmed by software when enabling and disabling a display. Programming is done through the DPLL_CTRL2 registers. See the section on Port Clock Programming.

Transcoder Clocks

There is one Transcoder clock tied to each display transcoder, except Transcoder WD which uses only CD clock.

	Transcoder clock EDP	Transcoder clocks A, B, C
Usage	Transcoder EDP symbol clock	Transcoders A, B, and C symbol/TMDS clocks
Input	Always uses DDI A clock	Programmable selection between DDI clocks B, C, D, and E.
Frequency	PLL output frequency divided by 5	PLL output frequency divided by 5
Default After Reset	Connected to DDI A Clock	Disabled

Programming	Not programmable.	Mapping of DDI to Transcoder must be programmed by software when enabling and disabling a display. Programming is done through the TRANS_CLK_SEL registers.

CD Clock

CD clock refers to the Core Display clock which includes the Core Display 1X Clock (CD clock, CDclk, cdclk, CDCLK) and the Core Display 2X Clock (CD2X clock, cd2xclk, CD2XCLK).

	CD clock
Usage	Clocking for most display engine functions
Input	DPLL0 CD2X output (with several frequency options) and Non-SSC reference.
Frequency	Non-SSC reference - 24 MHz CD, 24 MHz CD2X DPLL0 with VCO 8100: • 450 MHz CD, 900 MHz CD2X • 540 MHz CD, 1080 MHz CD2X • 337.5 MHz CD, 675 MHz CD2X • 675 MHz CD, 1350 MHz CD2X DPLL0 with VCO 8640: • 432 MHz CD, 864 MHz CD2X • 540 MHz CD, 1080 MHz CD2X • 308.57 MHz CD, 617.14 MHz CD2X • 617.14 MHz CD, 1234.28 MHz CD2X
Default After Reset	Enabled with the Non-SSC reference - 24 MHz CD, 24 MHz CD2X
Programming	CD clock cannot be disabled. CD clock frequency must be programmed by software when enabling and disabling a display. Programming is done through the CDCLK_CTL register.

Port Clock Programming

DisplayPort Programming

DisplayPort PLL Enable Sequence

1. Configure DPLL_CTRL1 to enable programming, set SSC enable/disable (no SSC for DPLL0), select DP mode, and set link rate.
2. Read back DPLL_CTRL1 to ensure writes completed before the next step.
3. Enable DPLL through LCPLL1_CTL (DPLL0), LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).
4. Wait for PLL lock status in DPLL_STATUS.
5. For each DDI that will use this DPLL. Configure DPLL_CTRL2 to turn on the clock for the DDI and map the DPLL to the DDI.

DisplayPort PLL Disable Sequence

1. For each DDI that was using this DPLL. Configure DPLL_CTRL2 to turn off the clock for the DDI.
2. Disable DPLL through LCPLL1_CTL (DPLL0), LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).

Example of DisplayPort on DDIA using HBR 2.7 GHz link rate with SSC

DPLL0 does not support SSC, so this case must use one of the other DPLLs. This example assumes DPLL1 is available.

1. Configure DPLL1 in the DPLL_CTRL1 register
   - DPLL1 Override = 1b (Enable)
   - DPLL1 Link Rate = 001b (1350 MHz - DP 2.7 GHz)
   - DPLL1 SSC = 1b (Enable)
   - DPLL1 HDMI Mode = 0b (DP mode)
2. Read back DPLL_CTRL1 to ensure writes completed
3. Enable DPLL1 in the LCPLL2_CTL register
   - PLL Enable = 1b (Enabled)
4. Wait for PLL lock status in DPLL_STATUS
5. Configure DPLL1 mapping to DDIA in the DPLL_CTRL2 register
   - DDIA Select Override = 1b (Enable)
   - DDIA Clock Select = 01b (DPLL1)
   - DDIA Clock Off = 0b (On)

HDMI and DVI Programming

HDMI and DVI PLL Enable Sequence

1. Configure DPLL_CTRL1 to enable programming, disable SSC, and use HDMI mode.
2. Configure DPLL_CFGCR1 to enable programming and set DCO frequency.
3. Configure DPLL_CFGCR2 to set the dividers and DCO central frequency.
4. Read back DPLL_CTRL1, DPLL_CFGCR1, and DPLL_CFGCR2 ensure writes completed before the next step.
5. Enable DPLL through LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).
6. Wait for PLL lock status in DPLL_STATUS.
7. For each DDI that will use this DPLL. Configure DPLL_CTRL2 to turn on the clock for the DDI and map the DPLL to the DDI.

HDMI and DVI PLL Disable Sequence

1. For each DDI that was using this DPLL. Configure DPLL_CTRL2 to turn off the clock for the DDI.
2. Disable DPLL through LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).

Formula for HDMI and DVI DPLL Programming

Reference frequency = 24 MHz.

Symbol clock frequency MHz = DCO Frequency / (P0 * P1 * P2) / 5.

AFE clock = Symbol clock frequency MHz * 5 = DCO Frequency / (P0 * P1 * P2).

P2 can be 1, 2, 3, or 5.

If P2 != 2, then P1 must be 1. Else P1 can be 1 to 255.

If P1 == 1, then P0 can be 1, 2, 3, or 7. Else P0 can be 2, 3, or 7.

An even value for the divider (P0 * P1 * P2) is preferred.

DCO central frequency must be 8400, 9000, or 9600 MHz.

DCO frequency must be within +1% or -6% of DCO central frequency. It is preferred to get as close to the DCO central frequency as possible, but using an even divider value takes precedence.

Note: If the desired symbol clock frequency cannot be achieved with the valid values of P0, P1, P2, and DCO frequencies, use a different screen resolution with a different symbol clock frequency.

Algorithm to Find HDMI and DVI DPLL Programming

1. Find AFE clock
2. For each legal divider (P*Q*K), find the DCO. Try even dividers first. Only use odd dividers if a good even is not found.

3. Try the DCO with each central frequency.
4. Select the divider and central frequency that gives the DCO with minimum deviation from DCO central frequency and fits within the +1% and -6% requirement.
5. Find the P, Q, K values to create that divider.

Pseudo-code for HDMI and DVI DPLL Programming

```
 Minimum Positive Deviation = 1%
 Minimum Negative Deviation = 6%
 Chosen Central Frequency = NONE
 AFE Clock = 5 * Symbol Clock MHz
 Even Candidate Dividers[] = { 4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 28, 30, 32, 36, 40, 42,
44, 48, 52, 54, 56, 60, 64, 66, 68, 70, 72, 76, 78, 80, 84, 88, 90, 92, 96, 98 }
 Odd Candidate Dividers[] = { 3, 5, 7, 9, 15, 21, 35 }

 For each Divider Parity (Even, Odd) // Prefer even values for divider
    If Divider Parity == Even
       Candidate Dividers[] = Even Candidate Dividers[]
    Else
       Candidate Dividers[] = Odd Candidate Dividers[]
    For each DCO Central Frequency // Try each central frequency
       For each Candidate Dividers // Try each divider
          DCO Frequency = Candidate Divider * AFE Clock
          If DCO Frequency > DCO Central Frequency  // Positive Deviation
             DCO Central Frequency Deviation = 100 * (DCO Frequency - DCO Central Frequency)
/ DCO Central Frequency
             If (DCO Central Frequency Deviation < Minimum Positive Deviation) // Check if it
meets the minimum requirement
                Minimum Positive Deviation = DCO Central Frequency Deviation
                Chosen Central Frequency = DCO Central Frequency
                Chosen DCO Frequency = DCO Frequency
                Chosen Divider = Candidate Divider
          Else // Negative Deviation
             DCO Central Frequency Deviation = 100 * ABS(DCO Frequency - DCO Central
Frequency)  / DCO Central Frequency
             If (DCO Central Frequency Deviation < Minimum Negative Deviation)
                Minimum Negative Deviation = DCO Central Frequency Deviation
                Chosen Central Frequency = DCO Central Frequency
                Chosen DCO Frequency = DCO Frequency
                Chosen Divider = Candidate Divider
       Next Candidate Divider
    Next DCO Central Frequency
    If Chosen Central Frequency != NONE // Break out early if a good even divider is found
       break
 Next Divider Parity
 getMultiplier(num)
 {
     if (num % 2 == 0) // Even
     {
         num1 = num / 2;
         if (num1 == 1 || num1 == 2 || num1 == 3 || num1 == 5)
         {
             P0 = 2;
             P1 = 1;
             P2 = num1;
         }
         else if (num1 % 2 == 0) // Div by 4
         {
             P0 = 2;
```

```
            P1 = num1 / 2;
            P2 = 2;
        }
        else if (num1 % 3 == 0) // Div by 6
        {
            P0 = 3;
            P1 = num1 / 3;
            P2 = 2;
        }
        else if (num1 % 7 == 0) // Div by 14
        {
            P0 = 7;
            P1 = num1 / 7;
            P2 = 2;
        }
    }
    // 3, 5, 7, 9, 15, 21, 35,
    else if (num == 3 || num == 9)
    {
        P0 = 3;
        P1 = 1;
        P2 = num / 3;
    }
    else if (num == 5 || num == 7)
    {
        P0 = num;
        P1 = 1;
        P2 = 1;
    }
    else if (num == 15)
    {
        P0 = 3;
        P1 = 1;
        P2 = 5;
    }
    else if (num == 21)
    {
        P0 = 7;
        P1 = 1;
        P2 = 3;
    }
    else if (num == 35)
    {
        P0 = 7;
        P1 = 1;
        P2 = 5;
    }
}
If (Chosen Central Frequency != NONE)
   getMultiplier(Chosen Divider)
   Program DPLL_CFGCR1 with Chosen DCO Frequency/24
   Program DPLL_CFGCR2 with Chosen DCO Central Frequency, P0 (P), P1 (Q), and P2 (K)
   Return Success // Valid result found
Return Fail  // No valid result found
```

Example of DVI on DDIB using 113.309 MHz symbol clock

This example assumes DPLL3 is available.

Frequency programming algorithm finds DCO central frequency = 9000 MHz, P0=2, P1=4, P2=2, DCO Frequency = 9064.72

1. Configure DPLL3 in the DPLL_CTRL1 register
    - DPLL3 Override = 1b (Enable)
    - DPLL3 SSC = 0b (Disable)
    - DPLL3 HDMI Mode = 1b (HDMI mode)
2. Configure DPLL3 in the DPLL3_CFGCR1 register

    DCO Integer = INT(9064.72/24) = 377

    DCO Fraction = INT(((9064.72/24) - DCO Integer) * 2^15) = 22,828

    - Frequency Enable = 1b (Enable)
    - DCO Fraction = 592Ch (22828)
    - DCO Integer = 179h (377)
3. Configure DPLL3 in the DPLL3_CFGCR2 register
    - Qdiv Ratio = P1 = 04h (4)
    - Qdiv Mode = 1b (Enable)
    - Kdiv = P2 = 01b (2)
    - Pdiv = P0 = 001b (2)
    - Central Frequency = 01b (9000 MHz)
4. Read back DPLL_CTL1, DPLL3_CFGCR1, and DPLL3_CFGCR2 to ensure writes completed
5. Enable DPLL3 in the WRPLL_CTL2 register
    - PLL Enable = 1b (Enabled)
6. Wait for PLL lock in DPLL_STATUS
7. Configure DPLL3 mapping to DDIB in the DPLL_CTRL2 register
    - DDIB Select Override = 1b (Enable)
    - DDIB Clock Select = 11b (DPLL3)
    - DDIB Clock Off = 0b (On)

Example of HDMI on DDIC using 296.703 MHz symbol clock

This example assumes DPLL2 is available.

Frequency programming algorithm finds DCO central frequency = 9000 MHz, P0=1, P1=3, P2=2, DCO Frequency = 8901.09

1. Configure DPLL2 in the DPLL_CTRL1 register
    - DPLL2 Override = 1b (Enable)
    - DPLL2 SSC = 0b (Disable)
    - DPLL2 HDMI Mode = 1b (HDMI mode)

2. Configure DPLL2 in the DPLL2_CFGCR1 register

   DCO Integer = INT(8901.09/24) = 370

   DCO Fraction = INT(((8901.09/24) - DCO Integer) * 2^15) = 28,794

   - Frequency Enable = 1b (Enable)
   - DCO Fraction = 707Ah (28794)
   - DCO Integer = 172h (370)
3. Configure DPLL2 in the DPLL2_CFGCR2 register
   - Qdiv Mode = 0b (Disable) // P1=1
   - Kdiv = P2 = 01b (2)
   - Pdiv = P0 = 010b (3)
   - Central Frequency = 01b (9000 MHz)
4. Read back DPLL_CTL1, DPLL2_CFGCR1, and DPLL2_CFGCR2 to ensure writes completed
5. Enable DPLL2 in the WRPLL_CTL1 register
   - PLL Enable = 1b (Enabled)
6. Wait for PLL lock status in DPLL_STATUS
7. Configure DPLL2 mapping to DDIC in the DPLL_CTRL2 register
   - DDIC Select Override = 1b (Enable)
   - DDIC Clock Select = 10b (DPLL2)
   - DDIC Clock Off = 0b (On)

Sequences for Changing CD Clock Frequency

The CD clock frequency can be programmed to select between several frequencies.

Restrictions
The CD clock frequency impacts the maximum supported pixel rate and display watermark programming. The CD clock frequency must be at least twice the frequency of the Azalia BCLK. Do not select the CD clock frequencies that have been restricted by the Display CDCLK Limit fuse (register DFSM).
Sequence for Changing CD Clock Frequency
1. Disable all display engine functions using the full mode set disable sequence on all pipes, ports, and planes. • Includes Global Time Code 2. Inform power controller of upcoming frequency change a. Ensure any previous GT Driver Mailbox transaction is complete. b. Write GT Driver Mailbox Data0 (GTTMMADDR offset 0x138128) = 0x00000003. c. Write GT Driver Mailbox Data1 (GTTMMADDR offset 0x13812C) = 0x00000000. d. Write GT Driver Mailbox Interface (GTTMMADDR offset 0x138124) = 0x80000007.

Restrictions
e. Poll GT Driver Mailbox Interface for Run/Busy indication cleared (bit 31 = 0). • Timeout after 150 us. Do not change CD clock frequency if there is a timeout. f. Read GT Driver Mailbox Data0, if bit 0 is 0x1, continue, else go to step b. • If the condition in step f is not satisfied after cycling through steps b-f for 3 ms (typically <200 us), timeout and do not change CD clock frequency. 3. Change CDCLK_CTL register CD Frequency Select and CD Frequency Decimal to the desired frequency • The frequency change will complete within a few clock cycles. 4. Inform power controller of the selected frequency a. Write GT Driver Mailbox Data0 with the frequency selection. • If selecting 337.5 or 308.57 MHz CD clock, or disabling CD clock PLL, write 0x00000000. • If selecting 450 or 432 MHz CD clock, write 0x00000001. • If selecting 540 MHz CD clock, write 0x00000002. • If selecting 675 or 617.14 MHz CD clock, write 0x00000003. b. Write GT Driver Mailbox Data1 = 0x00000000. c. Write GT Driver Mailbox Interface = 0x80000007. • The power controller should complete within 100 us, but there is no need for display software to wait for that. 5. Update programming of register fields that are based on CD clock frequency Programming can be delayed to when the features are enabled. • Utility pin backlight frequency and duty cycle in the BLC_PWM_DATA register.

Resets

NDE_RSTWRN_OPT

The north and south display engines are reset by PCI Function Level Resets (FLR) and the chip level resets.

An FLR for Bus:Device:Function 0:2:0 resets the north and south display engines and audio codec and most of the related MMIO, PCI, and I/O configuration registers.

Display configuration registers that are reset by both the chip level reset and by FLR are marked as using the "soft" reset in the programming specification.

Display configuration registers that are reset only by the chip level reset and *not* by FLR are marked as using the "global" reset in the programming specification.

The south display engine runs panel power down sequencing (if configured to do so) before resetting.

Shared Functions

Fuses and Straps

FUSE_STATUS

DFSM

DSSM

DFSDO NE

Interrupts

MASTER_INT_CTL

GT Interrupt 0 Definition

GT Interrupt 1 Definition

GT Interrupt 2 Definition

GT Interrupt 3 Definition

DE Pipe Interrupt Definition

DE Port Interrupt Definition

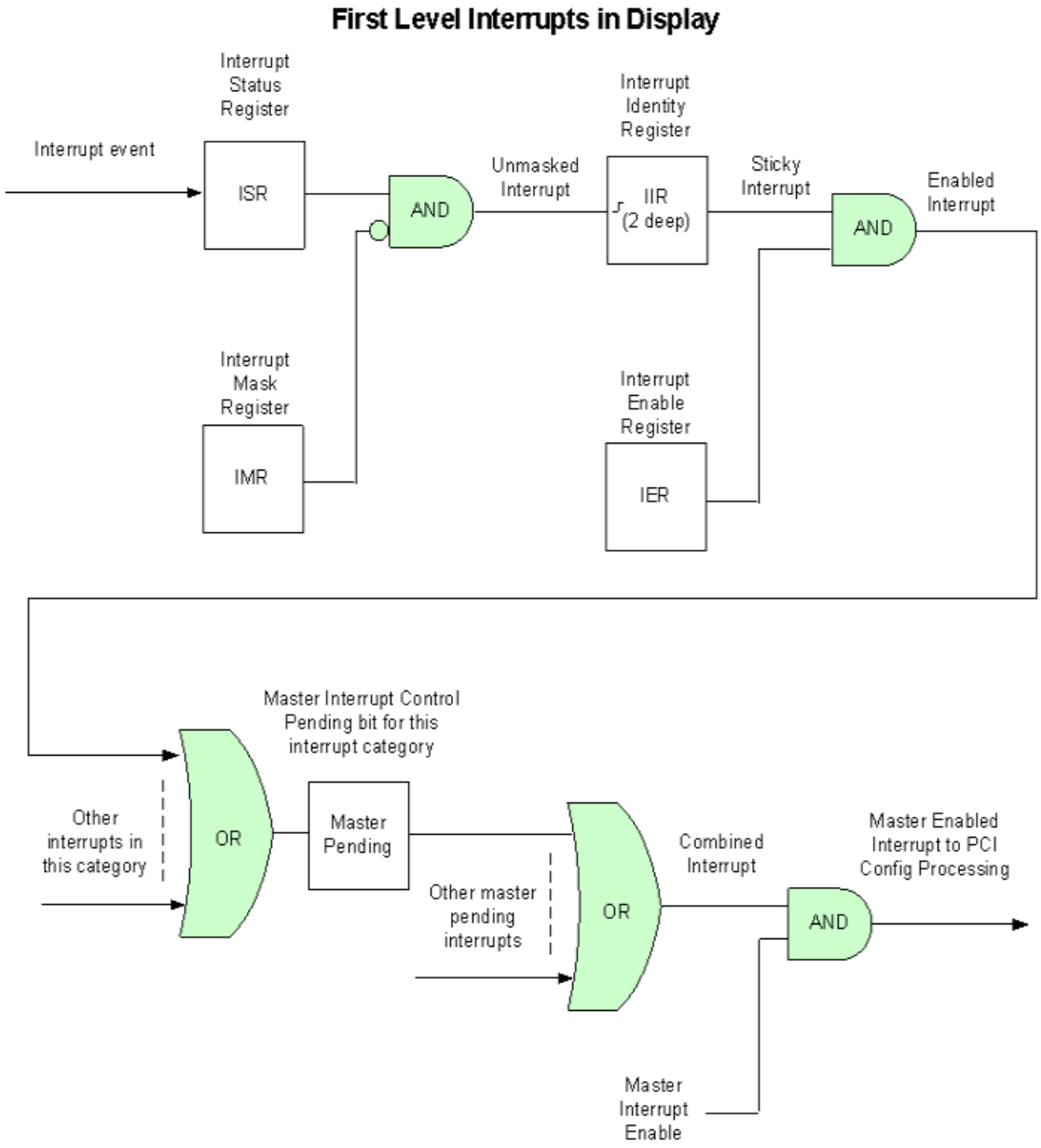
DE Misc Interrupt Definition

Audio Codec Interrupt Definition

PCU Interrupt Definition

INTERRUPT Structure

Interrupt Flow



1. For every first level interrupt bit
    a. The interrupt event comes in to the interrupt handling logic.
        - There may be more levels of interrupt handling behind each event. For example, the PCH Display interrupt event is the result of the SDE interrupt registers.
    b. The interrupt event goes to the Interrupt Status Register (ISR) where live status can be read back.
        - The live status is mainly useful for hotplug interrupts where it gives the live state of the hotplug line.
        - The live status is not useful for pulse interrupt events due to the short period that the status will be present.
    c. The interrupt event is ANDed with the inverted Interrupt Mask Register (IMR) to create the unmasked interrupt.
        - Only unmasked interrupts will proceed.
    d. The unmasked interrupt rising edge sets the sticky bit in the Interrupt Indentity Register (IIR).
        - The IIR can be cleared by writing a 1 to it.
        - The IIR can queue up to two interrupt events. When the IIR is cleared, it will set itself again if a second event was stored.
    e. The sticky interrupt is ANDed with the Interrupt Enable Register (IER) to create the enabled interrupt.
        - Only enabled interrupts will proceed.
2. All enabled interrupts are then ORed by category (Pipe, Audio Codec, Render, etc.) to create a combined interrupt which is then visible in one of the Master Interrupt Control Register (MASTER_INT_CTL) pending category bits.
3. All pending interrupts are then ORed to create the combined interrupt.
4. The combined interrupt is ANDed with the Master Interrupt Enable (MASTER_INT_CTL Bit 31) to create the master enabled interrupt.
    - Only a master enabled interrupt will proceed.
5. The master enabled interrupt then goes to PCI device 2 configuration registers PCISTS2, PCICMD2, and MC which control the MSI and line interrupt.

A Function Level Reset (FLR) or Warm Reset will reset all graphics interrupt logic, causing the master enabled interrupt to de-assert which can cause the MSI or line interrupt to de-assert.

Interrupt Service Routine

1. Disable Master Interrupt Control
    - Clear bit 31 of MASTER_INT_CTL (0x44200)
    - This is required to prevent missing any interrupts occurring back to back or during the service routine

2. Find the category of interrupt that is pending
    - Read MASTER_INT_CTL (0x44200) and record which interrupt pending category bits are set
3. Find the source(s) of the interrupt and clear the Interrupt Identity bits (IIR)
    - Read the IIR associated with each pending interrupt category, record which bits are set, then write back 1s to clear the bits that are set.
4. Process the interrupt(s) that had bits set in the IIRs
5. Re-enable Master Interrupt Control
    - Set bit 31 of MASTER_INT_CTL (0x44200)

Render Response

Display Engine Render Response Message Definition

DE_RRMR

DE_RR_DEST

Arbiter

Arbiter
ARB_CTL
ARB_CTL2

GSA

GTSP0

GTSP1 - Multiple Force Wake

GTSP2

GTSP3

GTSP4

GTSP5

GTSP6

GTSP7

Data Buffer

DBUF_CTL

DBUF_ECC_STAT

Backlight

This section refers to the CPU display backlight control. For PCH display backlight control, see South Display Engine Registers.

The backlight PWM output frequency is determined by the PWM clock frequency, increment, and frequency divider.

PWM output frequency = PWM clock frequency / PWM increment / PWM frequency divider

The frequency divider must be greater than or equal to the number of brightness levels required by software; typically 100 or 256.

Description
PWM clock frequency = CD clock frequency = 308.57 to 675 MHz PWM increment = 128 or 8, selectable by software PWM frequency divider maximum = 2^16 PWM output frequency range with PWM clock frequency 308.57 MHz and 100 brightness levels and increment 128 = 37 to 24,107 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 100 brightness levels and increment 8 = 589 to 385,713 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 256 brightness levels and increment 128 = 37 to 9,417 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 256 brightness levels and increment 8 = 589 to 150,669 Hz

Backlight Enabling Sequence

Description
1. Enable utility pin, select PWM mode, and set polarity in UTIL_PIN_CTL Util Pin Enable, Util Pin Mode, and Util Pin Output Polarity. 2. Set frequency and duty cycle in BLC_PWM_DATA Backlight Frequency and Backlight Duty Cycle. 3. Enable PWM output and set granularity in BLC_PWM_CTL PWM Enable and PWM granularity. ... 4. Change duty cycle as needed in BLC_PWM_DATA Backlight Duty Cycle.

If needed, granularity and polarity can be programmed earlier than shown.

Backlight Registers

Description
BLC_PWM_CTL
BLC_PWM_DATA

Miscellaneous Shared Functions

UTIL_PIN_CTL - Utility Pin Control

HDPORT_STATE

TOUCH_MSG_ADDR

DOUBLE_BUFFER_CTL

SWF

GTSCRATCH

Central Power

Frame Buffer Compression

FBC Registers

FBC_CFB_BASE

FBC_CTL

DPFC_CONTROL_SA

DPFC_CPU_FENCE_OFFSET

Workaround	
Context:	FBC
To prevent missed invalidations around the time FBC is being enabled, FBC render tracking must use the "Render Tracking With Nuke" method.	

FBC Overview

Frame Buffer Compression (FBC) gives a lossless compression of the display frame buffer to save power by reducing system memory read bandwidth and increasing the time between display engine reads to system memory.

FBC is only available on specific plane(s), depending on project. See FBC_CTL for details. FBC compresses pixels for the plane(s) it is attached to as they are displayed. The compressed data is written into the Compressed Frame Buffer (CFB) in graphics data stolen memory. The compressed data is then read the next time the same line needs to be displayed. Changes to the display front buffer (currently displayed

memory surface) through host aperture (GMADR) tiling fences (on projects that support that), render (RCS), and blitter (BCS) are tracked and cause the compressed lines to be invalidated and recompressed. Flips or changes to plane size and panning cause the entire buffer to be recompressed (nuke).

FBC Compression Limit

The FBC compression limit reduces the size of the Compressed Frame Buffer (CFB) by limiting which lines will be compressed. This is used when the graphics stolen memory available for the FBC CFB is smaller than the size of the original uncompressed frame buffer. There is also a FBC compressed vertical limit, listed in FBC_CTL, which is the maximum number of lines FBC can compress. Lines beyond the vertical limit do not need to be accounted for in the CFB size.

When the compression limit is 1:1, every line is written to the CFB, so the CFB width is the same as the original uncompressed frame buffer.

When the compression limit is 2:1, only lines that compress to 1/2 their original size will be written to the CFB, so the CFB width can be 1/2 the original uncompressed frame buffer.

When the compression limit is 4:1, only lines that compress to 1/4 their original size will be written to the CFB, so the CFB width can be 1/4 the original uncompressed frame buffer.

CFB size = ((Stride of plane uncompressed surface / FBC compression limit) * MIN(FBC compressed vertical limit, plane vertical source size)

FBC Programming Overview

1. Set up the compressed frame buffer.
    - The compressed buffer resides in graphics data stolen memory.
    - The stolen memory must be contiguous and un-cached.
    - The stolen memory needed for compressed frame buffer must be greater or equal to CFB size (calculation above).
    - Manage the compressed buffer size at run-time by balancing other graphics memory needs with the FBC allocation, and implement appropriate memory needs prioritization schemes.
2. Tracking for CPU host front buffer modifications through aperture fences (on projects that support that)
    - Setup System Agent (SA) registers to track CPU modifications on the display front buffer.
3. Tracking for display front buffer rendering
    - Setup Target Base Address, Front Buffer Target, and Address Valid for FBC
    - Enable Address Valid before rendering to the front buffer.
    - If needed, send nuke LRIs after each render submission to the display front buffer
4. Tracking from display front buffer BLTs
    - Setup Target Base Address and Address Valid for FBC
    - Enable Address Valid when BLTs target the display front buffer and disable when BLTs do not target the display front buffer

- Send nuke LRI or cache clean LRI after each blitter submission to the display front buffer

LRI commands to display address 0x50380 are used as part of the render and blitter tracking. Those LRIs must be followed SRM commands to the same address.

LRI to address 0x50380 with data 0x00000004 tells FBC to nuke and invalidate the entire compressed buffer.

LRI to address 0x50380 with data 0x00000002 tells FBC that previous blitter submissions have been flushed to memory and the invalidate lines can now be recompressed.

Render and blitter tracking can invalidate individual lines or nuke the entire buffer. The nuke method is used for workarounds.

Render Tracking With Nuke

- Software must send the nuke LRI after each render to the display front buffer

1. Render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
2. More render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
3. Render commands that do not touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL

Render Tracking Without Nuke

- Render tracking must be setup by software, then hardware will track FBC line invalidations automatically.

1. Render commands that potentially touch the display front buffer
    a. LRIs to FBC_RT_BASE_ADDR_REGISTER_<UPPER,LOWER> with display front buffer address, front buffer target=1, base address valid for FBC=1
    b. PIPE_CONTROL to ensure LRI takes effect
    c. Render submission
    d. PIPE_CONTROL

2. More render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
3. Render commands that do not touch the display front buffer
    a. Optional: LRIs to FBC_RT_BASE_ADDR_REGISTER_<UPPER,LOWER> with base address valid for FBC=0. Optional because hardware will detect that the render is not to the display front buffer address.
    b. Optional: PIPE_CONTROL to ensure LRI takes effect
    c. Render submission
    d. PIPE_CONTROL

Blitter Tracking With Nuke

- Software must send the nuke LRI after each BLT to the display front buffer. Never set 221d0h bit 3 (Address Valid for FBC).

1. BLT commands that touch the display front buffer
    a. BLT submission
    b. MI_FLUSH_DW
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
2. More BLT commands that touch the display front buffer (can leave Address Valid set from previous)
    a. BLT submission
    b. MI_FLUSH_DW
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
3. BLT commands that do not touch the display front buffer
    a. BLT submission
    b. MI_FLUSH_DW

Blitter Tracking Without Nuke

- Blitter tracking must be enabled by software only when BLTs touch the display front buffer. Software must send the cache clean LRI after each BLT to the display front buffer.

1. BLT commands that definitely touch the front buffer
    a. LRI to address 221d0h to set bit 3 (Address Valid for FBC)
    b. MI_FLUSH_DW to ensure LRI takes effect
    c. BLT submission

   d. MI_FLUSH_DW
   e. LRI to address 0x50380 with data 0x00000002 (cache clean)
   f. SRM to read address 0x50380 and store to a scratch page
2. More BLT commands that touch the front buffer (can leave Address Valid for FBC set from previous)
   a. BLT submission
   b. MI_FLUSH_DW
   c. LRI to address 0x50380 with data 0x00000002 (cache clean)
   d. SRM to read address 0x50380 and store to a scratch page
3. BLT commands that do not touch the front buffer (must clear Address Valid for FBC)
   a. LRI to address 221d0h to set bit 3 (Address Valid for FBC)
   b. MI_FLUSH_DW to ensure LRI takes effect
   c. BLT submission
   d. MI_FLUSH_DW

CPU Host Aperture Tracking

- Host tracking must be setup by software, then hardware will track FBC line invalidations to the specified tiling fence.

1. Program Y offset from the CPU fence to the Display Buffer base in DPFC_CPU_FENCE_OFFSET.
2. Program CPU fence ID and enable CPU fence tracking in DPFC_CONTROL_SA.

Disable CPU Fence tracking in DPFC_CONTROL_SA if no tiling fence is mapped to display buffer.

Display Plane Enabling with FBC

- FBC has to be enabled after the attached display plane is enabled.
- This is the general sequence. See FBC_CTL for any workarounds.

1. Enable display plane
2. Enable FBC as described in FBC_CTL register

Display Plane Disabling with FBC

- FBC has to be disabled prior to FBC attached display plane disabling.
- This is the general sequence. See FBC_CTL for any workarounds.

1. Disable FBC as described in FBC_CTL register
2. Disable display plane.

Watermarks

The watermark registers are used to control the display to memory request timing. The watermarks must be programmed according to the Display Watermark Programming section.

Description or Link
WM_PIPE, WM_LP, and WM_LP_SPR have been replaced by PLANE_WM and moved into the planes section.
WM_MISC **WM_LINETIME**

DC States

DC_STATE_EN

Power Wells

When a power well is disabled (powered down), access to any registers in the power well will complete, but write data will be dropped and read data will be all zeroes.

The power well enable requests from all sources are logically ORd together to enable a power well, so the power well will only disable after all sources have requested the power well to disable.

Register Link
PWR_WELL_CTL

Pipe

Color Space Conversion

CSC_COEFF

CSC COEFFICIENT FORMAT

CSC_PREOFF

CSC_POSTOFF

CSC_MODE

The high color channel is the most significant bits of the color. The low color channel is the least significant bits of the color. The medium color channel is the bits between high and low. For example: In RGB modes Red is in the High channel, Green in Medium, and Blue in Low. In YUV modes, V is in the High channel, Y in Medium, and U in Low.

The color space conversion registers are double buffered and are updated on the start of vertical blank following a write to the CSC Mode register for the respective pipe.

The matrix equations are as follows:

OutputHigh = (CoefficientRY * InputHigh) + (CoefficientGY * InputMedium) + (CoefficientBY * InputLow)

OutputMedium = (CoefficientRU * InputHigh) + (CoefficientGU * InputMedium) + (CoefficientBU * InputLow)

OutputLow = (CoefficientRV * InputHigh) + (CoefficientGV * InputMedium) + (CoefficientBV * InputLow)

Example programming for RGB to YUV is in the following table:

The input is RGB on high, medium, and low channels respectively and the desired YUV output is VYU on high, medium, and low channels respectively.

Program CSC_MODE to put gamma before CSC.

Program the CSC Post-Offsets to +1/2, +1/16, and +1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

	Bt.601		Bt.709	
	Value	Program	Value	Program
RU	0.2990	0x1990	0.21260	0x2D98
GU	0.5870	0x0968	0.71520	0x0B70
BU	0.1140	0x3E98	0.07220	0x3940
RV	-0.1687	0xAAC8	-0.11460	0xBEA8
GV	-0.3313	0x9A98	-0.38540	0x9C58
BV	0.5000	0x0800	0.50000	0x0800
RY	0.5000	0x0800	0.50000	0x0800
GY	-0.4187	0x9D68	-0.45420	0x9E88
BY	-0.0813	0xBA68	-0.04580	0xB5E0

Example programming for YUV to RGB is in the following table:

The input is VYU on high, medium, and low channels respectively.

The output is RGB on high, medium, and low channels respectively.

Program CSC_MODE to put gamma after CSC.

Program the CSC Pre-Offsets to -1/2, -1/16, and -1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

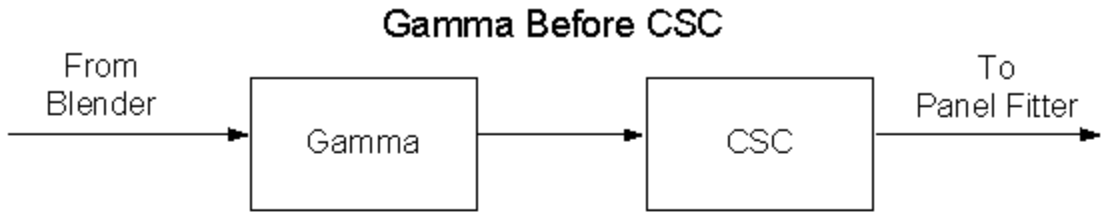
	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GY	1.000	0x7800	1.000	0x7800
BY	0.000	0x0000	0.000	0x0000
RY	1.371	0x7AF8	1.574	0x7C98
GU	1.000	0x7800	1.000	0x7800
BU	-0.336	0x9AC0	-0.187	0xABF8
RU	-0.698	0x8B28	-0.468	0x9EF8

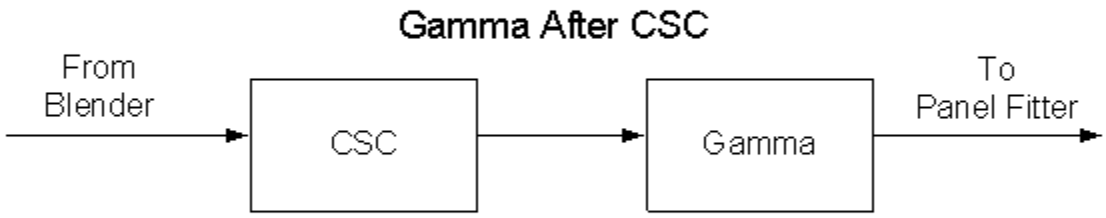
	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GV	1.000	0x7800	1.000	0x7800
BV	1.732	0x7DD8	1.855	0x7ED8
RV	0.000	0x0000	0.000	0x0000

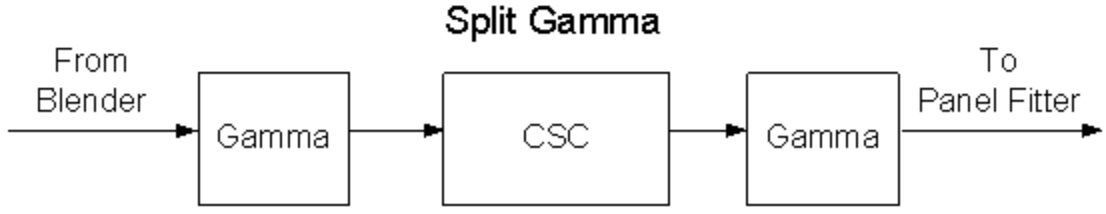
The pipe gamma and color space conversion blocks can be placed in three different arrangements:

- Gamma before CSC: Selected through the CSC Mode register. Mostly used for RGB to YUV conversion.
- Gamma after CSC: Selected through the CSC Mode register. Mostly used for YUV to RGB conversion or linear RGB to RGB conversion. Can be used with pipe color gamut enhancement.
- Split gamma: Selected through the Gamma Mode register. Mostly used for RGB to RGB conversion. Can be used with pipe color gamut enhancement. The pipe gamma enable per plane will control whether a plane will go through both gamma blocks. It is not possible to send a plane through one gamma block and not the other.

In either arrangement, the final output of the pipe gamma and CSC and gamut enhancement logic is clamped to fit in the 0 to 1.0 range before going to the ports.







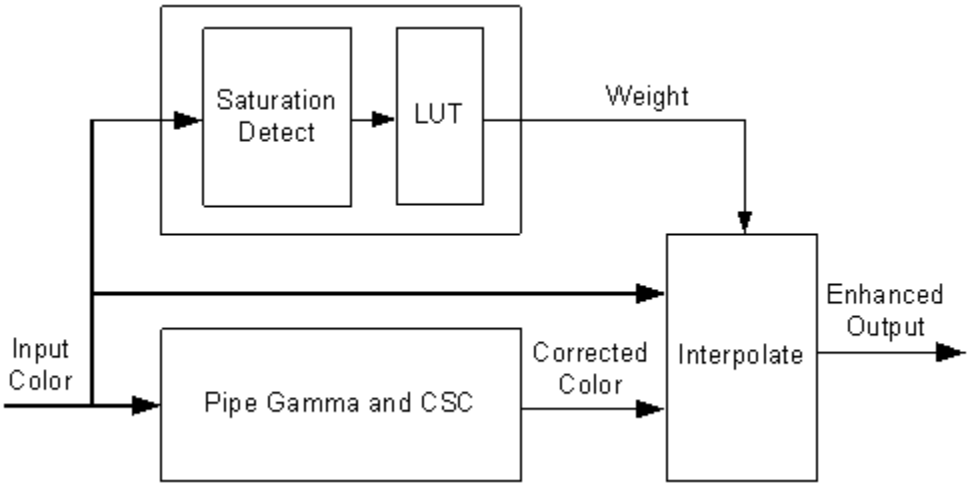
Pipe Color Gamut Enhancement

Pipe color gamut enhancement is used to enhance display of standard gamut content on wide gamut displays. It processes the color value from before and after the pipe gamma and color space correction blocks to create the color gamut enhanced output. The typical usage is to output the pipe gamma and CSC corrected color for areas of low saturated content and the input (not gamma or CSC corrected) color for areas of high saturated content. It is not recommended to use color gamut enhancement with wide gamut inputs.

CGE_CTRL

CGE_WEIGHT

The pipe Gamma and CSC must be programmed to either the split gamma mode or gamma after CSC mode when using pipe color gamut enhancement.



The saturation level of the pipe gamma and CSC input color is detected and used to index into a look up table (LUT) containing programmable weights. The saturation values are linearly distributed across the LUT indexes from the lowest index for lowest saturation to the highest index for highest saturation.

The enhanced output color is created by using the weight value to interpolate between the input color and corrected color. See the following table of weights to amount of input or corrected color used to create the enhanced output color.

Weighting of input and corrected colors

Weight from LUT	Amount of Input Color in Enhanced Output	Amount of Corrected Color in Enhanced Output
00 0000b (minimum)	0%	100%
...	...	...
00 1000b	25%	75%
...	...	...
01 0000b	50%	50%
...	...	...
01 1000b	75%	25%
...	...	...
10 0000b (maximum)	100%	0%

Example weight programming

CGE LUT Index	CGE Weight Value Decimal	CGE Weight Value Binary	CGE Weight Percent Input Color	CGE Weight Percent Corrected Color
0 (lowest saturation)	0	00 0000b	0%	100%
1	0	00 0000b	0%	100%
2	0	00 0000b	0%	100%
3	0	00 0000b	0%	100%
4	0	00 0000b	0%	100%
5	0	00 0000b	0%	100%
6	1.6	00 0010b	5%	95%
7	3.2	00 0011b	10%	90%
8	4.8	00 0101b	15%	85%
9	6.4	00 0110b	20%	80%
10	8.64	00 1001b	27%	73%
11	12.8	00 1101b	40%	60%
12	19.2	01 0011b	60%	40%
13	25.6	01 1010b	80%	20%
14	28.8	01 1101b	90%	10%
15	32	10 0000b	100%	0%
16 (highest saturation)	32	10 0000b	100%	0%

Pipe DPST

DPST_CTL

DPST_BIN

DPST_GUARD

Pipe Palette and Gamma

The display palette provides a means to correct the gamma of an image stored in a frame buffer to match the gamma of the monitor or presentation device. Additionally, the display palette provide a method for converting indexed data values to color values for VGA and 8-bpp indexed display modes. The display palette is located after the plane blender. Using the individual plane gamma enables, the blended pixels can go through or bypass the palette on a pixel by pixel basis.

PAL_LGC

PAL_PREC_INDEX

PAL_PREC_DATA

PAL_GC_MAX

PAL_EXT_GC_MAX

GAMMA_MODE

If any gamma value to be programmed exceeds the maximum allowable value in the associated gamma register, then the programmed value must be clamped to the maximum allowable value.

Programming Modes

The display palette can be accessed through multiple methods and operate in one of four different modes as follows.

8 bit legacy palette/gamma mode:

This provides a palette mode for indexed pixel data formats (VGA and primary plane 8 bpp) and gamma correction for legacy programming requirements.

All input values are clamped to the 0.0 to 1.0 range before the palette/gamma calculation. It is not recommended to use legacy palette mode with extended range formats.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the 256 palette/gamma entries. The 256 entries are stored in the legacy palette with 8 bits per color in a 0.8 format with 0 integer and 8 fractional bits.

The legacy palette is programmable through both MMIO and VGA I/O registers. Through VGA I/O, the palette can look as though there are only 6 bits per color component, depending on programming of other VGA I/O registers.

10 bit gamma mode:

This provides the highest quality gamma for pixel data formats of 30 bits per pixel or less.

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 1024 gamma entries. The first 1024 entries are stored in the precision palette with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 1024th and 1025th gamma entries to create the result value. The 1025th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Split gamma mode:

Split gamma mode is composed of two gamma functions. The first gamma is before pipe color space conversion (CSC) and the second is after CSC. This split gamma mode permits mapping to linear gamma, then color space conversion, then mapping to monitor gamma. This provides the highest quality pipe color space conversion and gamma correction for inputs with non-linear gamma.

First gamma (before CSC):

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 512 gamma entries. The first 512 entries are stored in the precision palette indexes 0 to 511 with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 512th and 513th gamma entries to create the result value. The 513th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Second gamma (after CSC):

All input values are clamped to the 0.0 to 1.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 512 gamma entries. The first 512 entries are stored in the precision palette indexes 512 to 1023 with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

12 bit interpolated gamma mode:

This provides the highest quality gamma for pixel data formats greater than 30 bits per pixel.

The gamma correction curve is represented by specifying a set of gamma entry reference points spaced equally along the curve for values between -1 and 1. For extended values there is an extended gamma entry reference point at the maximum allowed input value.

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

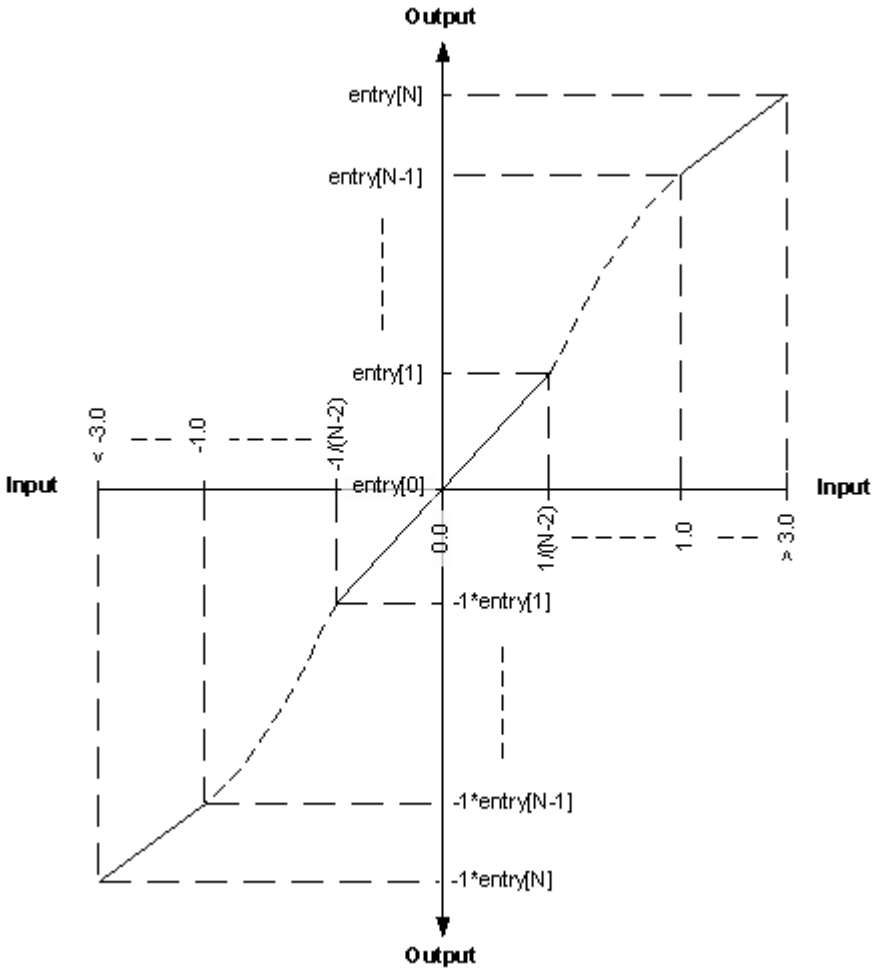
For input values greater than or equal to 0 and less than 1.0, the input value is used to linearly interpolate between two adjacent points of the first 513 gamma entries to create the result value. The first 512 entries are stored in the precision palette with 16 bits per color in a 0.16 format with 0 integer and 16 fractional bits (upper 10 bits in odd indexes, lower 6 bits in even indexes). The 513th entry is stored in the PAL_GC_MAX register with 17 bits per color in a 1.16 format with 1 integer and 16 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 513th and 514th gamma entries to create the result value. The 514th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

To program the gamma correction entries, calculate the desired gamma curve for inputs from 0 to 3.0. The curve must be flat or increasing, never decreasing. For inputs of 0 to 1.0, multiply the input value by 512 to find the gamma entry number, then store the desired gamma result in that entry. For inputs greater than 1.0 and less than or equal to 3.0, store the result for an input of 3.0 in the 514th gamma entry.

Example Pipe Gamma Correction Curve



Pipe Control

PIPE_SRCSZ

PIPE_SCANLINE

PIPE_SCANLINECOMP

PIPE_MISC

PIPE_BOTTOM_COLOR

PIPE_FRMCNT

PIPE_FLIPCNT

PIPE_FRMTMSTMP

PIPE_FLIPTMSTMP

Pipe Scaler

Each scaler has its own set of registers.

PS_WIN_POS

PS_WIN_SZ

PS_CTRL

PS_VSCALE

PS_HSCALE

PS_VPHASE

PS_HPHASE

PS_ECC_STAT

Planes

Plane Capability and Interoperability

Plane Assignments and Capabilities

Plane Capabilities:

	Pipe A	Pipe B	Pipe C
Plane 3	Plane or Cursor	Plane or Cursor	Plane or Cursor
Plane 2	Plane + Render Decompression	Plane + Render Decompression	Plane
Plane 1	Plane + Render Decompression, Frame Buffer compression	Plane + Render Decompression	Plane

Mapping to Command Streamer Plane Number

Display Plane Name	Command Streamer Plane Number
Plane 1 A	Plane 1
Plane 1 B	Plane 2
Plane 1 C	Plane 3
Plane 2 A	Plane 4
Plane 2 B	Plane 5
Plane 2 C	Plane 6
Plane 3 A	Plane 7
Plane 3 B	Plane 8
Plane 3 C	Plane 9
N/A	Plane 10
N/A	Plane 11
N/A	Plane 12

Plane Feature Interoperability

Display Features / Surface Formats:

	RGB32 2:10:10:10	RGB32 8:8:8:8	RGB 32-bit XR_BIAS 10:10:10	RGB64 16:16:16:16	RGB 16-bit 5:6:5	Indexed 8-bit	YUV 32-biwt 4:4:4	YUV 16-bit 4:2:2
Linear - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
Linear - 90/270 rotation								
X Tiling - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
X Tiling - 90/270 rotation								
Y Tiling (legacy) - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
Y Tiling (legacy) - 90/270 rotation	✓	✓	✓				✓	✓
Yf Tiling - 0/180 rotation	✓	✓	✓		✓		✓	✓
Yf Tiling - 90/270 rotation	✓	✓	✓				✓	✓
Frame Buffer Compression		✓			✓			
Render Compression		✓						
Plane Scaling	✓	✓			✓		✓	✓
Per-pixel Alpha		✓						
Color Keying	✓	✓	✓	✓	✓	✓	✓	✓

Tiling Modes / Display Features:

	Rotation 0/180	Rotation 90/270	Render Decompression	Frame Buffer Compression	Interlacing (Interlaced Fetch)
Linear	✓			✓	✓
X Tiling	✓			✓	✓
Y Tiling Legacy	✓	✓ (some surface formats not supported)	✓	✓	
Yf Tiling	✓ (some surface formats not supported)	✓ (some surface formats not supported)	✓	✓	

Rotation Modes / Display Features:

	Interlacing	Frame Buffer Compression	Render Decompression
0/180 rotation	✓	✓	✓
90/270 rotation		✓	

Render/Display Decompression

GT Render engines uses a lossless scheme to compress the color Render targets. The goal of the compression is to reduce the memory bandwidth. The memory foot print increases slightly due to the need of the control surface.

- Decompression is supported on plane 1 and plane 2 of pipe A and pipe B with RGB8888 surface formats.
- Decompression is supported only in legacy tile Y and tile Yf surfaces.
- Decompression support is limited to left-right cache line pair mode. Top-bottom mode is not supported.
- When render decompression is enabled, display engine internally converts the Async flips to Sync flips.
- Decompression is not supported with 90/270 degree rotation.

Color Control Surface

The Color Control Surface (CCS) contains the compression status of the cache-line pairs**.** The compression state of the cache-line pair is specified by 2 bits in the CCS. Each CCS cache-line represents an area on the main surface of 16 x16 sets of 128 byte Y-tiled cache-line-pairs. CCS is always Y tiled. The

address of CCS surface is specified as an offset from the start of the Render Target main surface. CCS stride is programmed separately independent of the main surface stride.

Decompression Programming

When compressed Render targets are presented to Display, the display decompression must be enabled. Along with main surface programming, the following additional programming is required to enable the decompression

- **Decompression Enable**

Decompression is enabled by programming the 'Render Decomp' bit in the PLANE_CTL register.

- **Color Control Surface Distance**

The start of the CCS surface is programmed as the distance in bytes from the start of the main surface in the PLANE_AUX_DIST register. The CCS is always placed after the main surface and is 4K page aligned.

- **Color Control Surface Stride**

The CCS stride is programmed in the PLANE_AUX_DIST register.

Plane Rotation Programming

This topic provides programming information for plane rotation.

The 180 rotation mode is unchanged and will continue to use the same programming. In the 180 rotation mode display hardware is responsible for walking the pages in the reverse order. The cacheline walk within the page is also reversed. The 90 and 270 rotation modes require more complicated page walk mechanism. The page walk is made transparent to the hardware by providing a different set of page translations (remapped GTT) for the same rendered surface. The remapping completely abstracts the page walk away from the hardware and the hardware walks the pages as if there is no rotation. Hardware is still responsible for handling the the walk within the page appropriately. Also, the 90, 270 Rotation requires a new parameter - surface height. The changes needed in the driver programming is discussed below.

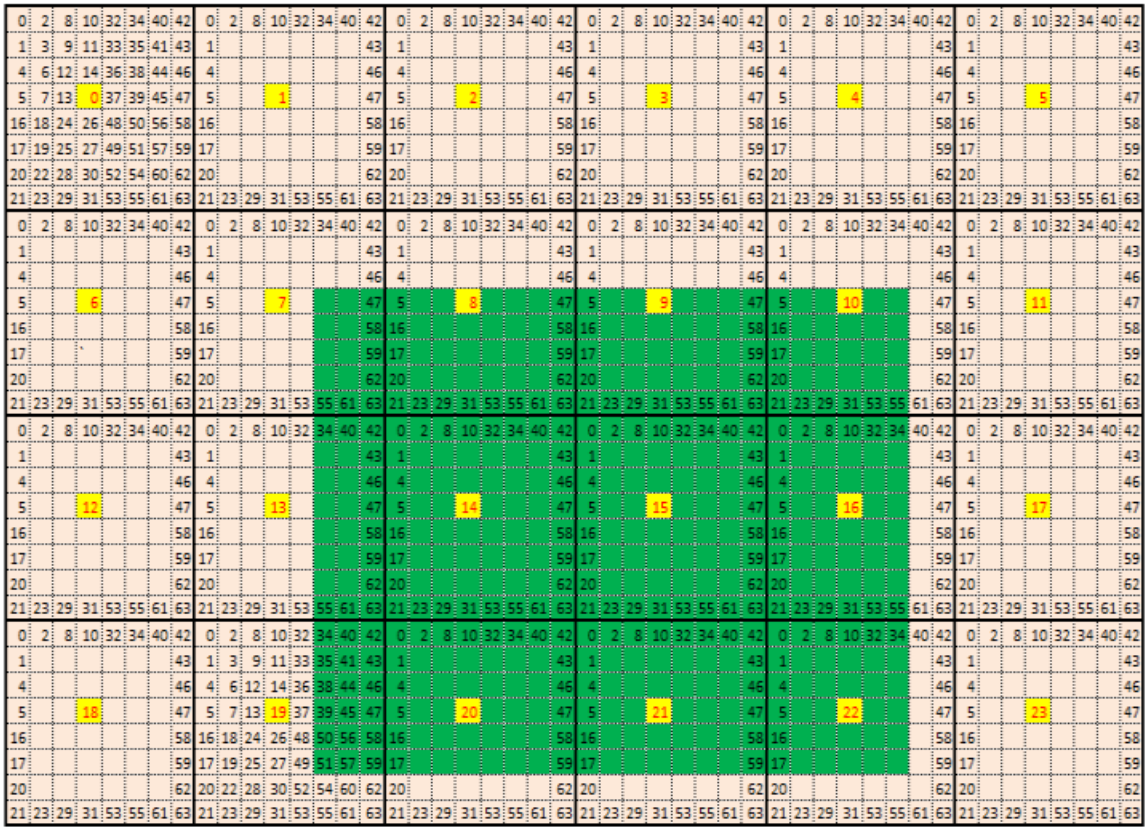
90 Rotation

For the 90 rotation programming, the plane parameters must use the following mapping

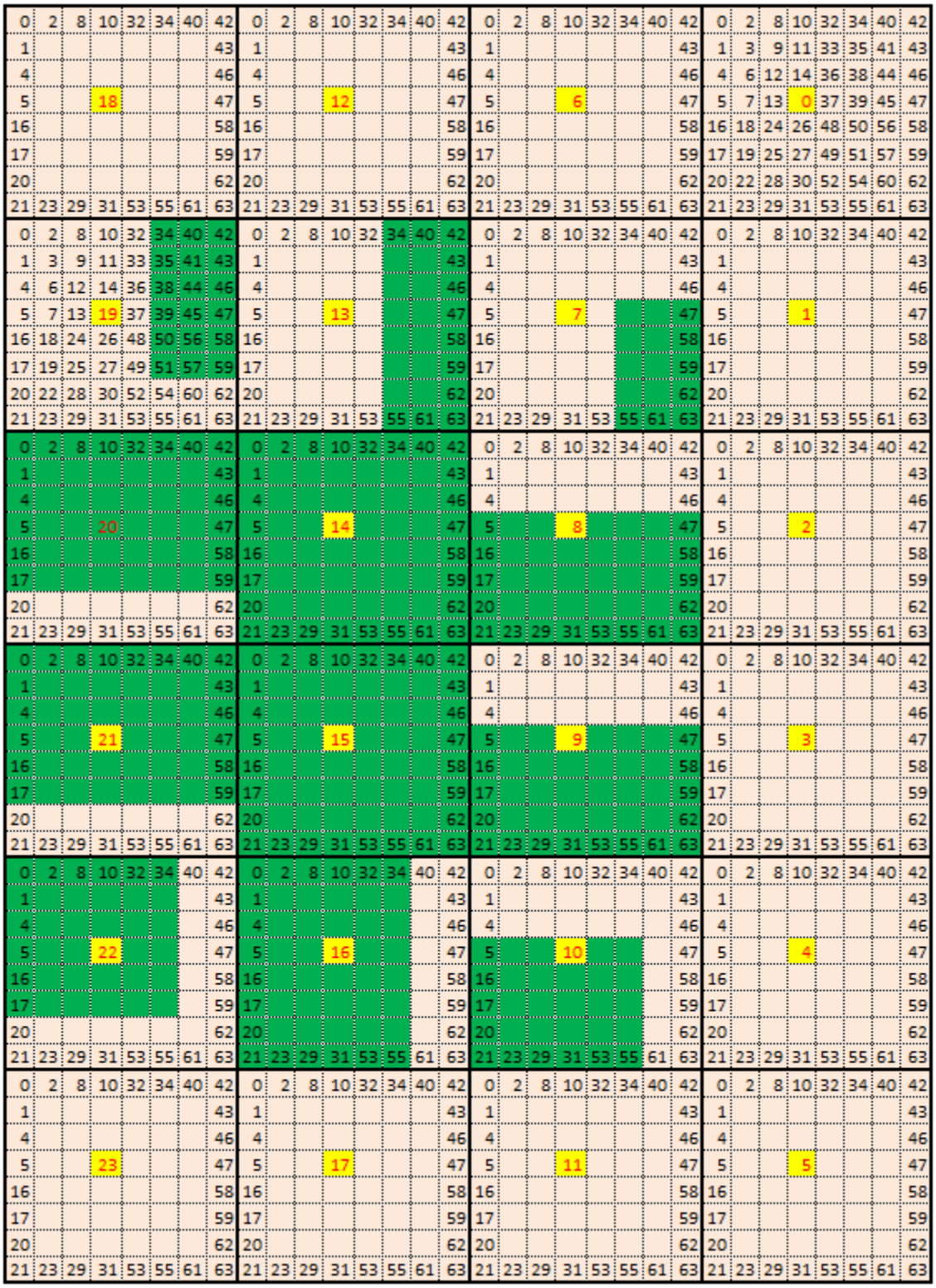
- Base address = New address (remapped GTT)
- Stride = Surface height in tiles
- Plane Width = Plane Height
- Plane Height = Plane Width
- X offset = (Surface height in tiles * tile height) - Plane Y offset - Plane Height *[Note: the calculated X offset will always be >= 0 since Plane Y Offset + Plane Height <= Surface Height in lines]*

- Y offset = X offset
- Driver can enable the Scaler, as needed, to fit the rotated content in to the pipe active display area.

Sample surface in frame buffer (Yf tiled)



Page access sequence in the rotated 90 mode



Example

Let us assume the following display programming for a single pipe – single plane, Yf tiled, non-rotated, 1920 x 1200, 4Bpp with the plane panned (100, 150), covering full active area and scaler not enabled.

GTT mapping

Here is a sample GTT mapping for 90 rotation mode.

Original GTT

Assumed Surface base = 0x200000

0x200000 = page 0

0x201000 = page 1

0x202000 = page 2

0x203000 = page 3

…

Remapped Display GTT – 90 rotation

Assumed new Surface base = 0x400000

0x400000 = page 18

0x401000 = page 12

0x402000 = page 6

0x403000 = page 0

Register programming for non-rotated scenario

- *PLANE_SURF->Surface Base Address = 0x200000*
- *PLANE_STRIDE->Stride = 60* [(1920 * 4)/128] [(width *bpp)/tile width]
- *PLANE_SIZE->Width = 1920*
- *PLANE_SIZE->Height = 1200*
- *PLANE_OFFSET->Start X Position = 100*
- *PLANE_OFFSET->Start Y Position = 150*
- *Surface Height in tiles (assumed) = 50 (allocated surface height in number of scan lines/tile height. For plane height = 1200, the surface height should be a minimum of 38 tiles (ceiling (1200/32). When panning is used, the rendered frame buffer surface will be larger than the plane size. Here, let us assume that the rendered surface height in tiles = 50).*

The programming changes to following for a 90 rotation scenario

- *PLANE_SURF->Surface Base Address = 0x400000* [uses remapped GTT]
- *PLANE_STRIDE->Stride = 50* [Surface height in tiles (assumed earlier)]
- *PLANE_SIZE->Width = 1200* [non-rotated Height]

- *PLANE_SIZE->Height* = *1920* [non-rotated Width]
- *PLANE_OFFSET->Start X Position* = *250* [(50*32)-150-1200] [(Surface height * tile height) – non rotated Y position – non rotated Height]
- *PLANE_OFFSET->Start Y Position* = *100* [non-rotated X position]

The scaler should be programmed appropriately to fit the rotated plane in the pipe active area and the window position should be adjusted if it is desired to maintain the same apparent position on a physically rotated display.

270 rotation

Uses the same GTT remapping and register programming as 90 rotation mode with the Plane control register rotation mode set as 270.

Display Buffer Programming

This topic describes display buffer allocation and shows a basic allocation method for single and multi-pipe modes. The display driver can choose to use more advanced allocation techniques as desired.

Display Buffer Allocation

Allocation of the display buffer is programmable for each display plane, using the buffer start and buffer end values in PLANE_BUF_CFG_.

Proper display buffer allocation is important for Display hardware to function correctly. Optimal allocation provides better display residency in memory low power modes. Display Buffer allocation must be recalculated and programmed when pipes/planes get enabled or disabled.

Display Buffer Size

Display Buffer Size	Total Display Buffer Blocks	Fixed Bypass Path Allocation in Blocks	Blocks Available for Driver Programming
448 KB	896	4	0 - 891

Each display buffer block is 8 cache lines.

Allocation Requirements

Allocation must not overlap between any enabled planes.

A minimum allocation is required for any enabled plane. See Minimum Allocation Requirements below.

A gap between allocation for enabled planes is allowed.

The allocation for enabled planes should be as large as possible to allow for higher watermarks and better residency in memory power saving modes.

Minimum Allocation Requirements

Allocation for each enabled plane must meet these minimum requirements.

Planes using Linear or X tiled memory formats must allocate a minimum of 8 blocks.

Planes using Y tiled memory formats must allocate blocks for a minimum number of scanlines worth of data. The formula and table of minimum scanlines is below.

Y tiled minimum allocation = Ceiling [(4 * Plane source width * Plane Bpp)/512] * MinScanLines/4 + 3

	Minimum Scanlines for Y Tile	
Plane Bpp	0/180 Rotation	90/270 Rotation
1	8	32
2	8	16
4	8	8
8	8	4

Basic Allocation Method

These are basic methods that can be used for single and multi-pipe modes. For optimal power usage, the display driver can choose to use more advanced allocation techniques as desired.

Single Pipe

Allocate a fixed number of blocks to cursors and then allocate the remaining blocks among planes, based on each plane's data rate.

BlocksAvailable = TotalBlocksAvailable

1. Allocate 32 blocks for cursor

The driver frequently enables and disables the cursor or changes the cursor pixel format. Fixed allocation is preferred for cursor to minimize the buffer re-allocation. More allocation might be required to support deeper low power states (based on the results of watermark calculations).

CursorBufAlloc = 32

BlocksAvailable = BlocksAvailable – 32

2. Check for minimum buffer requirement

For each enabled plane

If Y tiled

MinScanLines = Look up minimum scanlines needed from the table

*PlaneMinAlloc = Ceiling [(4*Plane width*Bpp)/512] * MinScanLines/4 + 3*

Else

PlaneMinAlloc = 8

If sum of PlaneMinAlloc > BlocksAvailable

Error - Display Mode can't be supported.

The driver can change the number of enabled planes or the plane configuration and rerun the algorithm.

3. Calculate Relative Data Rate for planes

In this step the driver may want to use the expected maximum plane source sizes so it does not have to reallocate for a plane that is changing size.

For each enabled plane

If PlaneScalerEnabled

*PlaneScaleFactor = (Plane width/Scaler window X size) * (Plane height/Scaler window Y size)*

Else

PlaneScaleFactor = 1

*PlaneRelativeDataRate = Plane height * Plane width * plane source bytes per pixel * PlaneScaleFactor*

4. Allocate blocks for enabled planes as per the Data rate ratio.

For each plane that needs allocation (PlaneBlockAllocFinal == false)

*PlaneBufAlloc = floor (BlocksAvailable * PlaneRelativeDataRate/Sum of PlaneRelativeDataRate of all planes that need allocation).*

*floor - rounds down to an integer value dropping the fractional part.

5. Adjust for minimum allocation requirement

AdjustmentRequired = false

For each plane needs allocation (PlaneBlockAllocFinal == false)

If PlaneBufAlloc < PlaneMinAlloc

AdjustmentRequired = true

PlaneBufAlloc = PlaneMinAlloc

PlaneBlockAllocFinal = true

BlocksAvailable = BlocksAvailable - PlaneMinAlloc

If AdjustmentRequired = true

Go back to step 4

Multi-Pipe

NumPipes = Total number of display pipes in the hardware

Allocate a fixed number of blocks to cursors, allocate *1/NumPipes* of the remaining blocks to each pipe, then calculate each pipe individually as in the Single Pipe case.

1. Allocate 8 blocks for cursor per pipe

The driver frequently enables and disables the cursor or changes the cursor pixel format. Fixed allocation is preferred for cursor to minimize the buffer re-allocation. More allocation might be required to support deeper low power states (based on the results of watermark calculations).

For each enabled cursor

CursorBufAlloc = 8

*BlocksAvailable = TotalBlocksAvailable - (8 * NumPipes)*

2. Distribute the blocks equally among the pipes

   BlocksAvailable = BlocksAvailable/NumPipes

3. Assign blocks to the planes

   For each pipe

Perform Single Pipe sequence, starting from step 2.

Buffer allocation re-distribution

When an additional pipe is getting enabled, or an existing pipe requires more buffer to support a new mode or is disabled, buffer reallocation may be necessary for proper display functionality. Whenever a portion of the allocated buffer is taken away from one pipe and allocated to a different pipe, the following sequence should be followed to make sure that there are no buffer allocation overlaps at any point of time.

1. *For each pipe whose allocation is reduced*
   a. *Program the new buffer allocation.*
   b. *Wait for VBlank of that pipe for new allocation to update.*
2. *For each pipe whose allocation is increased*
   a. *Program the new buffer allocation.*
   b. *Wait for VBlank of that pipe for new allocation to update.*

Display Buffer Allocation and Watermark programming prior to OS boot

Basic programming of the display buffer and watermarks to allow limited display usage prior to OS boot:

This will prevent package power saving states from enabling.

Supported usages:

- Up to 3 pipes enabled at once
- Up to one universal plane enabled per pipe. No cursor.
- Linear or Xtile memory
- Any RGB frame buffer pixel format 32bpp or less, without render compression
- Any screen resolution
- Downscaling less than or equal to 12.5%

Allocate 160 blocks per pipe.

Pipe A: 0-159, Pipe B: 160-319, Pipe C: 320-479

PLANE_BUF_CFG_<plane number>_A = 0x009F0000

PLANE_BUF_CFG_<plane number>_B = 0x013F00A0

PLANE_BUF_CFG_<plane number>_C = 0x01DF0140

Set level 0 watermarks for any enabled plane to 160 blocks and 2 lines.

PLANE_WM_<plane number>_<pipe>_0 = 0x800080A0

The higher level watermarks for any enabled plane must have bit 31=0 to keep the low power watermarks disabled.

VGA

The VGA Control register is located here. The VGA I/O registers are located in the VGA Registers document.

VGA_CONTROL

Cursor Plane

Planes
CUR_CTL
CUR_BASE
CUR_POS
CUR_PAL
CUR_FBC_CTL
PLANE_SURFLIVE
CUR_SURFLIVE
PLANE_BUF_CFG
PLANE_WM

The CUR_CTL and CUR_FBC_CTL active registers will be updated on the vertical blank or when pipe is disabled, after the CUR_BASE register is written, or when cursor is not yet enabled, providing an atomic update of those registers together with the CUR_BASE register.

Universal Plane

PLANE_CTL

PLANE_STRIDE

PLANE_POS

PLANE_SIZE

PLANE_SURF

PLANE_LEFT_SURF

PLANE_AUX_DIST

PLANE_OFFSET

PLANE_KEYVAL

PLANE_KEYMSK

PLANE_KEYMAX

PLANE_SURFLIVE

PLANE_WM

PLANE_BUF_CFG

PLANE_GAMC

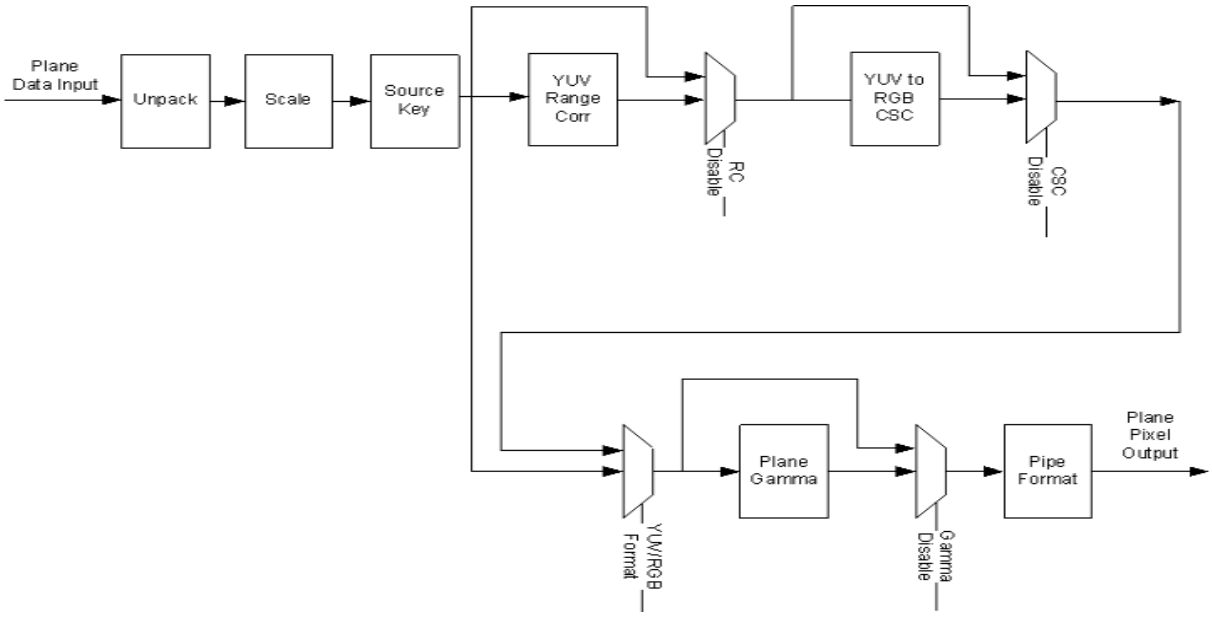
PLANE_GAMC16

PLANE_GAMC17

Many of the plane control active registers will be updated on the vertical blank or when pipe is disabled, after the surface base address register is written, or when the plane is not yet enabled, providing an atomic update of those registers together with the surface base address register.

Data flow through the plane (Steps 2-6 may be enabled or disabled by programming control bits):

1. Unpack data into pixels
2. Plane Scaling
3. Source Key
4. YUV Range Correction (can only be used by YUV source pixel formats)
5. YUV to RGB Color Space Conversion (can only be used by YUV source pixel formats)
6. Gamma Correction
7. Conversion to pipe data format



Plane Pixel Formats

Plane Source Pixel Format Mapping of Bits to Colors:

XRGB Formats	Ignored	Red	Green	Blue
RGB 32-bit 2:10:10:10 BGRX	31:30	29:20	19:10	9:0
RGB 32-bit 2:10:10:10 RGBX	31:30	9:0	19:10	29:20
RGB 32-bit 8:8:8:8 BGRX	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBX	31:24	7:0	15:8	23:16
RGB 64-bit 16:16:16:16 Float BGRX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 Float RGBX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	15:0	31:16	47:32
RGB 32-bit XR_BIAS 10:10:10	31:30	9:0	19:10	29:20
16-bit BGR 5:6:5	N/A	15:11	10:5	4:0

ARGB Formats	Alpha	Red	Green	Blue
RGB 32-bit 8:8:8:8 BGRA	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBA	31:24	7:0	15:8	23:16

XRGB Formats	Ignored	Red	Green	Blue
RGB 32-bit 2:10:10:10 BGRX	31:30	29:20	19:10	9:0
RGB 32-bit 2:10:10:10 RGBX	31:30	9:0	19:10	29:20
RGB 32-bit 8:8:8:8 BGRX	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBX	31:24	7:0	15:8	23:16
RGB 64-bit 16:16:16:16 Float BGRX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 Float RGBX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	15:0	31:16	47:32
RGB 32-bit XR_BIAS 10:10:10	31:30	9:0	19:10	29:20
16-bit BGR 5:6:5	N/A	15:11	10:5	4:0

ARGB Formats	Alpha	Red	Green	Blue
RGB 32-bit 8:8:8:8 BGRA	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBA	31:24	7:0	15:8	23:16

Transcoder

Transcoder Control

TRANS_CONF

TRANS_STEREO3D_CTL

Transcoder Timing

TRANS_HTOTAL

TRANS_HBLANK

TRANS_HSYNC

TRANS_VTOTAL

TRANS_VBLANK

TRANS_VSYNC

TRANS_SPACE

TRANS_VSYNCSHIFT

TRANS_MULT

Transcoder MN Values

Description
These values are used for DisplayPort.

There is one instance of these registers per each transcoder.

For dynamic switching between multiple refresh rates, M/N values may be reprogrammed on the fly. The link N should be programmed last to trigger the update of all the data and link M and N registers and then the new M/N values will be used in the next frame that is output.

DATAM

DATAN

LINKM

LINKN

Clocks:

ls_clk is the link symbol clock. i.e. 270 MHz for HBR2.

strm_clk is the stream clock, which is the video pixel rate or dot clock.

cdclk is the core display clock.

The link only operates in Synchronous Clock mode. The link clock and stream clock are synchronous and the link M and N values stay constant for a given pixel rate.

Calculation of TU:

TU is the Transfer Unit.

TU size calculation (applies to SST only)

TU size for reduced blanking cases may vary and the following formula should be used in such cases.

TU size = MIN(INT((Horizontal Blank Width * ls_clk/strm_clk)-2), 64)

Example 1:

ls_clk = 810 MHz

strm_clk = 1025.766 MHz

Horizontal Blank = 80 pixels (Reduced Blanking v2 timings)

Calculated TU size = MIN(INT(63.1-2),64) = 61

Actual TU size = 61

So, in this example TU size becomes 61.

Example 2:

ls_clk = 810 MHz

strm_clk = 923.221 MHz

Horizontal Blank = 80 pixels (Reduced Blanking v2 timings)

TU size = MIN(INT(70.18-2),64) = 64

Calculation of Data M, and Data N:

Note that for 4:2:0 format the number of bytes per pixel will be half the number of bytes of RGB pixel.

Active/TU Size = Payload/Capacity = Data M/N

Compression Ratio (CR) = DisplayPort Compression enabled ? min(ratio1,ratio2) : 1

- ratio1 = Compressor BW / Link BW = (cdclk * bytes per pixel) / (ls_clk * number of lanes)
- ratio2 = (Horizontal active in pixels * bytes per pixel / 4) / ((Horizontal active in pixels * bytes per pixel / 8) +2)

Data M/N = (strm_clk * bytes per pixel) / (CR * ls_clk * number of lanes)

Calculation of Link M and Link N:

Link M/N = strm_clk / ls_clk

Restriction on clocks and number of lanes:

- Number of lanes >= INT(strm_clk * bytes per pixel / ls_clk)

Restrictions on the Virtual Channel (VC) payload size in DisplayPort MST mode

- In a x1 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 4.

- In a x2 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 2.
- In a x4 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 1.

Transcoder Video Data Island Packet

Data Island Packet (DIP) is a mechanism that allows data to be sent over a digital port during blanking, according to the HDMI and DisplayPort specifications. This includes header, payload, checksum, and ECC information.

Each type of Video DIP will be sent once each frame while it is enabled.

VIDEO_DIP_CTL

VIDEO_DIP_DATA

VIDEO_DIP_GCP

VIDEO_DIP_ECC

Construction of DIP for AVI, VS, or SPD (HDMI only):

Dword	Byte3	Byte2	Byte1	Byte0
0	Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8 (RO)	Reserved	Reserved	Reserved	HB ECC
9 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0

HB = Header Byte, DB = Data Byte, RO = Read Only

Construction of DIP for GMP (HDMI or DisplayPort):

Dword	Byte3	Byte2	Byte1	Byte0
0	DP: HB3 HDMI: Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20

Dword	Byte3	Byte2	Byte1	Byte0
7	DB27	DB26	DB25	DB24
8 (RO)	Reserved	Reserved	Reserved	DP: Reserved HDMI: HB ECC
9 (RO)	DP: Reserved HDMI: DB ECC 3	DP: Reserved HDMI: DB ECC 2	DP: Reserved HDMI: DB ECC 1	DP: Reserved HDMI: DB ECC 0
10 (RO)	DP: HB ECC 3 HDMI: Reserved	DP: HB ECC 2 HDMI: Reserved	DP: HB ECC 1 HDMI: Reserved	DP: HB ECC 0 HDMI: Reserved
11 (RO)	DP: DB ECC 3 HDMI: Reserved	DP: DB ECC 2 HDMI: Reserved	DP: DB ECC 1 HDMI: Reserved	DP: DB ECC 0 HDMI: Reserved
12 (RO)	DP: DB ECC 7 HDMI: Reserved	DP: DB ECC 6 HDMI: Reserved	DP: DB ECC 5 HDMI: Reserved	DP: DB ECC 4 HDMI: Reserved

HB = Header Byte, DB = Data Byte, DP = DisplayPort, RO = Read Only

Construction of DIP for VSC (DisplayPort only):

Dword	Byte3	Byte2	Byte1	Byte0
0	HB3	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8	DB31	DB30	DB29	DB28
9 (RO)	HB ECC 3	HB ECC 2	HB ECC 1	HB ECC 0
10 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0
11 (RO)	DB ECC 7	DB ECC 6	DB ECC 5	DB ECC 4

HB = Header Byte, DB = Data Byte, RO = Read Only

The audio subsystem is also capable of sending Data Island Packets. These packets are programmed by the audio driver and can be read by in MMIO space via the audio control state and audio HDMI widget data island registers.

Video DIP data write sequence:

1. Wait for 1 VSync to ensure completion of any pending video DIP transmissions
2. Disable the video DIP being updated (disable VDSC before updating PPS DIP)
3. Program video DIP data buffer registers for DIP being updated
4. Enable the video DIP

The video DIP data and ECC buffers may be read at any time.

DIP data buffer registers must be programmed with valid data before enabling the DIP.

Partial DIPs are never sent out while the port is enabled. Disabling the DIP at the same time it is being transferred will result in the DIP being completed before the function is disabled.

Shutting off the port on which DIP is being transmitted will result in partial transfer of DIP data. There is no need to switch off the DIP enable bit if the port transmitting DIP is disabled.

When disabling both the DIP port and DIP transmission, first disable the port and then disable DIP.

Enabling a DIP function at the same time that the DIP would have been sent out (had it already been enabled) will result in the DIP being sent on the following frame.

For HDMI, even if no DIP is enabled, a single Null DIP will be sent at the same point in the stream that DIP packets would have been sent.

Transcoder DDI Function

TRANS_DDI_FUNC_CTL

TRANS_MSA_MISC

Panel Self Refresh

This section is about Panel Self Refresh (PSR). PSR1 at one point was named Self Refreshing Display (SRD).

PSR1 enable sequence:

- Prerequisite: The associated transcoder and port are running.

1. Configure FBC host and render tracking. The FBC function does not need to be enabled in FBC_CTL.
2. Program Transcoder EDP VSC DIP data with a valid setting for SRD/PSR.
3. Configure and enable SRD_CTL.

PSR1 disable sequence:

- Prerequisite: The associated transcoder and port are running.

1. Disable SRD_CTL.
2. Wait for SRD_STATUS to show SRD is Idle. This will take up to one full frame time (1/refresh rate), plus SRD exit training time (max of 6ms), plus SRD aux channel handshake (max of 1.5ms).

PSR2 disable sequence:

1. Program PSR2_CTL reset Psr2Enable, SelectiveUpdateTrackingEnable bits.
2. Disable GTC if required.
3. Wait for PSR2_STATUS to show PSR2 is Idle. This will take up to one full frame time (1/refresh rate), plus exit training time (max of 6ms), plus aux channel handshake (max of 1.5ms).

Registers
SRD_CTL
SRD_STATUS
SRD_PERF_CNT
SRD Interrupt Bit Definition
SRD_IMR
SRD_IIR
PSR_MASK
PSR_EVENT
PSR2_CTL
PSR2_MAN_TRK_CTL
PSR2_SU_STATUS

Registers
PSR2_STATUS

Transcoder Port Sync

Feature Description

PORT SYNC is a transcoder level feature. This mode forces two or more transcoders to be in sync with one transcoder master and one or more transcoder slaves. In the case of DP/eDP, the master is unaware that it is operating in Port Sync mode. Only the slave is aware that it is operating in this mode. Hence, port sync mode is only enabled in the slave transcoder.

DP/eDP Port Sync Restrictions

1. The slave and master transcoders and associated ports must have identical parameters and properties.
2. They must be connected to the same PLL, have the same color format, link width (number of lanes enabled), resolution, refresh rate, dot clock, TU size, M and N programming, etc.
3. Port Sync Mode must only be enabled with DisplayPort SST.
4. PSR would need to be disabled when port sync mode is enabled.
5. Port Sync Mode Master Select must be programmed with a valid value when Port sync Mode is enabled.

Audio

This section describes Audio features.

Audio Bios Programming Sequence

Codec Verb Table

For each codec present on the High Definition Audio codec link, a corresponding pre-defined “Codec Verb Table” must be available to System BIOS. The Codec Verb Tables are based on codec specific information (coded datasheet) and platform design specific information (schematics) and are built by System BIOS writers and platform designers. The table contains a list of 32-bit “Verb”s (command and data payload) to be sent to the corresponding codec over the High Definition Audio codec link.

Below is a sample High Definition Audio Codec Verb Table for a platform with 1 codec at codec address 01h.

;Sample HIGH DEFINITION AUDIO Codec Verb Table

;Codec Address (CAd) = 02h

;Codec Vendor: XYZ Company

;VenID DevID:

```
    dd 12345678h
;-----------------------------------------------------
; FrontPanel_Supported?   ; 1=Supported ,0=Not supported
    db  01h
; # of Rear Panel Pin Complexes
   dw 000Ch
; # of Front Panel Pin Complexes
dw 0002h
;-----------------------------------------------------
```

Note: Set the bit 15 of register offset 0x65F10h of the Display Audio offset. Wait for the Codec to generate the wake event to the controller.

Follwing verbs should be send to the codec using the PIO method described in the below sections 9.1.3.

VerbTable0:

```
    ;Enable the third converter and Pin first (NID 08h)
     dd 20878101h
       //
// Audio Verb Table - 0x80862805
//
// Pin Widget 5 - PORT B
0x20571C10,
0x20571D00,
0x20571E56,
0x20571F18,
// Pin Widget 6 - PORT C
0x20671C20,
0x20671D00,
0x20671E56,
0x20671F18,
// Pin Widget 7 - PORT D
0x20771C30,
0x20771D00,
```

0x20771E56,

0x20771F18,

// Pin Widget 8 - PORT E -

0x20871C40,

0x20871D00,

0x20871E56,

0x20871F18

;disable the third converter and third Pin (NID 08h - upto gen9)

dd 20878100h

Codec Initialization Programming Sequence

After System BIOS has determined the presence of High Definition Audio codecs, it must follow the programming sequence below to update the codec with the correct jack information specific to the platform for the High Definition Audio driver to retrieve and use later.

There are two ways to send verbs to and receive response data from codecs over the High Definition Audio codec link: using CORB/RIRB (Command Output Ring Buffer / Response Input Ring Buffer) or using the Immediate Command/Immediate Response register pair. The sequence below uses the latter which does not require the availability of a memory buffer.

System BIOS should ensure that the High Definition Audio HDBAR D27:F0:10-17h contains a valid address value and is enabled by setting D27:F0:04h[1]. System BIOS must ensure program as mentioned in section 9.6, and then the Controller Reset# bit of Global Control register in memory-mapped space (HDBAR+08h[0]) is set to 1b and read back as 1b. Additional delay might be required to allow codec coming out of reset prior to subsequent operations, please contact your codec vendor for detail. When clearing this bit and setting it afterward, BIOS must ensure that minimum link timing requirements (minimum RESET# assertion time, etc.) are met.

Note: To initialize codec Bios should set the bit 15 of the register 0X65F10h of the Display MMIO to 1. This bit needs to be after the controller is brought out of reset. BIOS should wait for Controller to detect the wake event and recognize the Codec.

For each High Definition Audio codec present as indicated by HDBAR + 0Eh[3:0], System BIOS should perform the codec initialization as described below:

1. Read the VendorID/DeviceID pair from the attached codec.

- Verify that the ICB bit, HDBAR + 68h[0], is 0.
- Write verb 200F0000h (dword) to the IC register, HDBAR + 60h, where: ‘2’ (bits 31:28) represents the codec address (CAd).
- Program HDBAR + 68h[1:0] to 11b to send the verb to the codec.

- Poll the ICB bit, HDBAR+68h[0] until it returns 0 indicating the verb has been sent to the codec. BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- If HDBAR + 68h[1] = 1b indicating the response data from the codec is now valid, read HDBAR + 64h; the data is the VID/DID value returned by the codec.

2. Check against internal list to determine if there is a stored verb table which matches the CAd/VID/DID information.

Steps 1 and 2 are System BIOS implementation-specific steps and can be done in different ways. If a System BIOS has prior knowledge of a fixed platform/codec combination (e.g., for a System BIOS having 3 stored verb tables for 3 known codecs at known codec addresses on a known platform), a simple pre-defined codec-to-table matching can be used and steps 1 and 2 can be eliminated. For a System BIOS to support multiple codec/platform combinations, an internal match-list might be needed to match a platform/codec combination to a codec verb table.

3. If there is a match, send the entire list of verbs in the matching verb table one by one to the codec.

- Verify the ICB bit, HDBAR + 68h[0] is 0.
- Write the next verb (dword) in the table to HDBAR + 60h.
- Program HDBAR + 68h[1:0] to 11b to send the verb to codec.
- Poll the ICB bit, HDBAR + 68h[0] until it returns 0 indicating the verb has been sent to the codec. BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- Repeat the steps until all the verbs in the table have been sent.

Some verbs in the table may be dependent on certain platform-specific conditions. For example, for the sample table above, the verbs for Pin Complex 7 and 8 (NID=14,16 respectively) should be sent only if the Front Panel Jacks are present and connected on the platform, which may be indicated by a software flag that is controlled by a certain GPIO pin.

Audio Programming Sequence for Link wakeup

The following audio programming sequences are to be used for preventing the Unsolicited responses when 3 pin link is awake.

Display Audio codec generates a wake event whenever the power well (PGx - power well in which Display Audio codec HW resides) is powered up. If the link is already running, this wake event is considered as unsolicited response by audio controller in PCH. This may sometimes be considered as unnecessary URs. To avoid such URs following programming should be followed by SW. This sequence assumes communication between Audio and GFX drivers without HW to indicate Audio codec power well status.

Power down sequence:

1. Unplug event
2. PD goes low

3. PG2 low

4. Audio link should go low (there should be a communication between GFX driver and Audio Driver to turn off the link)

    1. To turn off iDisp-A link:
    2. Power off iDisp-A codec. Follow the PW2 turn off sequence.
    3. In HD Audio Controller (PCH) Program LCTL1.SPA = 0.
    4. Wait for LCTL1.CPA = 0 to indicate the link has clock stopped.

Power up sequence:

1. Plug event

2. PG2 goes high

3. Audio link to be enabled (there should be a communication between GFX driver and Audio Driver to turn on the link)

    1. To turn on iDisp-A link:
    2. In HD Audio Controller (PCH) Program LCTL1.SPA = 1.
    3. Wait for LCTL1.CPA = 1 to indicate the link has clock running.

4. Check WAKESTS[2] = 1 to indicate the codec wake up occurs

5. PD bit set

6. Codec awake. Continue with codec init.

Audio Programming Sequence

The following HDMI and DisplayPort audio programming sequences are to be used when enabling or disabling audio or temporarily disabling audio during a display mode set.

The audio codec and audio controller disable sequences must be followed prior to disabling the transcoder or port in a display mode set.

The audio codec and controller enable sequences can be followed after the transcoder is enabled and the port is enabled and completed link training (not sending training or idle patterns if DisplayPort).

The audio controller and audio codec sequences may be done in parallel or serial. In general, the change in ELDV/PD in the codec sequence will generate an unsolicited response to the audio controller driver to indicate that the controller sequence should start, but other mechanisms may be used. SW should make sure to set the Inactive (IA) bit to 0 before setting PD to 1.

Audio codec disable sequence:

- Disable sample fabrication

    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to "0".
- Disable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "1"
    - Set Upper_N_value and Lower_N_value (bits 27:20, 15:4) to all "0"s.
- Disable ELDV and ELD buffer
    - Set AUD_PIN_ELD_CP_VLD ELD_valid (bit 0, 4, or 8 based on which port is used) to "0"
- Wait for 2 vertical blanks
- Optional: Disable audio PD (Presence Detect)
    - Software may choose to skip this in order to keep PD enabled during a resolution switch.
    - Set AUD_PIN_ELD_CP_VLD Audio_Inactive (bit 3, 7, or 11) to "1". SW does not need to set this bit to enable Inactive bit.
    - Set AUD_PIN_ELD_CP_VLD Audio_Output_Enable (bit 2, 6, or 10) to "0".

Audio controller disable sequence:

- Program Stream ID to 0 - Verb ID 706
- Disable audio info frames transmission - Verb ID 732
- Disable Digen - Verb ID 70D
- Program the codec to D3 state if needed.
- Audio driver may stop the audio controller DMA engine at this point if needed, but not required.

Audio codec enable sequence:

- Enable audio Presence Detect
    - Set AUD_PIN_ELD_CP_VLD Audio_Inactive (bit 3, 7, or 11) to "0".
    - Set AUD_PIN_ELD_CP_VLD Audio_Output_Enable (bit 2, 6, or 10) to "1".
- Wait for 1 vertical blank
- Load ELD buffer and Enable ELDV
    - Set AUD_PIN_ELD_CP_VLD ELD_valid (bit 0, 4, or 8 based on which port is used) to "1".
- Enable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "0".
    - Program Upper_N_value and Lower_N_value (bits 27:20, 15:4) if a non-default N value is needed.
- Enable sample fabrication if this feature is needed
    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to "1".

Audio controller enable sequence:

- Program the codec to D0 state if in D3 state.

- Program Stream ID to non zero - Verb ID 706
- Enable audio info frames transmission - Verb ID 732
- Enable Digen - Verb ID 70D
- If audio controller DMA engine is stopped, audio driver can start the DMA engine at this point.

Audio Configuration

AUD_CONFIG

AUD_MISC_CTRL

AUD_VID_DID

AUD_RID

AUD_M_CTS_ENABLE

Audio Power State Format

AUD_PWRST

AUD_EDID_DATA

AUD_FREQ_CNTRL

AUD_INFOFR

AUD_PIN_PIPE_CONN_ENTRY_LNGTH

AUD_PIPE_CONN_SEL_CTRL

AUD_DIP_ELD_CTRL_ST

AUD_PIN_ELD_CP_VLD

Digital Display Interface

DDI Buffer

There is one instance of these registers per each DDI.

DDI_BUF_CTL

DDI_BUF_TRANS

DISPIO_CR_TX_BMU_CR0

DDI_BUF_TRANS and DISPIO_CR_TX_BMU_CR0 are programmed with the voltage swing values for each port. They can be configured in advance of the display mode set. The recommended values are listed below.

For DisplayPort, DDI_BUF_CTL is programmed during the mode set to select between the voltage swings pre-programmed in DDI_BUF_TRANS.

HDMI always uses the voltage swing programmed in DDI_BUF_TRANS entry 9.

I_boost

I_boost increases the I/O current, boosting the swing level. There are three I_boost values, in order of increasing boost; 0x1, 0x3, and 0x7.

I_boost can be configured separately for each port and can be enabled or disabled separately for each voltage swing level.

I_boost is configured by programming DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=<I_boost value; 0x1, 0x3, or 0x7>. It is then enabled when DDI_BUF_CTL selects a voltage swing entry that has DDI_BUF_TRANS_<entry> Balance Leg Enable=0x1. For DDIA with x4 capability (DDI_BUF_CTL DDIA Lane Capability Control = DDIA x4), the I_boost value has to be programmed in both tx_blnclegsctl_0 and tx_blnclegsctl_4.

The recommended buffer translation programming only uses I_boost for a few entries. A customer may request increased I_boost beyond the recommended values by specifying an I_boost value of 0x1, 0x3, or 0x7 to be applied to all swing entries for a port, then software needs to configure I_boost to that specified value and set DDI_BUF_TRANS_<all entries> Balance Leg Enable=0x1 for that port. This selection overrides the recommended I_boost settings.

Programming Note	
Context:	DDI Buffer
KBL re-uses SKL values, except where specific KBL values are listed.	

Recommended Buffer Translation Programming

KBL - Recommended buffer translation programming for DisplayPort											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level**[1]	**Pre-emphasis Level**[1]	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**KBL H and S**		**KBL Y**		**KBL U**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	400	400	0	0000 00A0h	0000 2016h	0000 00A1h	0000 1017h	0000 00A1h	0000 201Bh
1	0	1	400	600	3.5	0000 009Bh	0000 5012h	0000 0088h	0000 5012h	0000 0088h	0000 5012h
2	0	2	400	800	6	0000 0088h	0000 7011h	0000 00**CD** h	**8**000 7011h [3]	0000 00**CD** h	**8**000 7011h [3]
3	0	3	400	1000	9.5	0000 00C0h	**8**000 9010h [2]	0000 00C0h	**8**000 800Fh [3]	0000 00C0h	**8**000 9010h [3]
4	1	0	600	600	0	0000 009Bh	0000 2016h	0000 009Dh	0000 1017h	0000 009Dh	0000 201Bh

KBL - Recommended buffer translation programming for DisplayPort											
5	1	1	600	900	3.5	0000 0088h	0000 5012h	0000 00C0h	**8**000 5012h [3]	0000 00C0h	**8**000 5012h [3]
6	1	2	600	1000	6	0000 00C0h	**8**000 7011h [2]	0000 00C0h	**8**000 7011h [3]	0000 00C0h	**8**000 7011h [3]
7	2	0	800	800	0	0000 0097h	0000 2016h	0000 004Ch	0000 1017h	0000 004Fh	0000 2016h
8	2	1	800	1000	3.5	0000 00C0h	**8**000 5012h [2]	0000 00C0h	**8**000 5012h [3]	0000 00C0h	**8**000 5012h [3]

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

Recommended buffer translation programming for embedded DisplayPorts that support low voltage swings											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level[1]**	**Pre-emphasis Level[1]**	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**SKL/KBL H and S**		**SKL/KBL Y**		**SKL/KBL U**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	200	200	0	0000 00A8h	0000 0018h	0000 00A8h	0000 0018h	0000 00A8h	0000 0018h
1	0	1	200	250	1.5	0000 00A9h	0000 4013h	0000 00ABh	0000 4013h	0000 00A9h	0000 4013h
2	0	2	200	350	6	0000 00A2h	0000 7011h	0000 00A4h	0000 7011h	0000 00A2h	0000 7011h
3	0	3	200	600	9.5	0000 009Ch	0000 9010h	0000 00DFh	0000 9010h	0000 009Ch	0000 9010h
4	1	0	250	250	0	0000 00A9h	0000 0018h	0000 00AAh	0000 0018h	0000 00A9h	0000 0018h
5	1	1	250	350	3.5	0000 00A2h	0000 6013h	0000 00A4h	0000 6013h	0000 00A2h	0000 6013h
6	1	2	250	500	6	0000 00A6h	0000 7011h	0000 009Dh	0000 7011h	0000 00A6h	0000 7011h
7	2	0	350	350	0	0000 00ABh	0000 0018h	0000 00A0h	0000 0018h	0000 00ABh	0000 2016h
8	2	1	350	500	4.5	0000 009Fh	0000 7013h	0000 00DFh	0000 6012h	0000 009Fh	0000 5013h

Recommended buffer translation programming for embedded DisplayPorts that support low voltage swings											
9	Fail Safe	Fail Safe	800	800	0	0000 00DFh	0000 0018h	0000 008Ah	0000 0018h	0000 00DFh	0000 0018h

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

Recommended buffer translation programming for HDMI and DVI						
Non-Transition mV diff p-p	**Transition mV diff p-p**	**Pre-emphasis dB**	**SKL/KBL U, H, and S**		**SKL/KBL Y**	
			Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
400	400	0	0000 00ACh	0000 0018h	0000 00A1h	0000 0018h
400	600	3.5	0000 009Dh	0000 5012h	0000 00DFh	0000 5012h
400	800	6	0000 0088h	0000 7011h	0000 00CBh	8000 7011h[3]
450	450	0	0000 00A1h	0000 0018h	0000 00A4h	0000 0018h
600	600	0	0000 0098h	0000 0018h	0000 009Dh	0000 0018h
600	800	2.5	0000 0088h	0000 4013h	0000 0080h	0000 4013h
600	1000	4.5	0000 00CDh	8000 6012h[2]	0000 00C0h	8000 6013h[3]
800	800	0	0000 00DFh	0000 0018h	0000 008Ah	0000 0018h
[1]800	1000	2	0000 00CDh	8000 3015h[2]	0000 00C0h	8000 3015h[3]
1000	1250	2	0000 00C0h	8000 3015h[2]	0000 00C0h	**8**000 3015h[3]
1200	1200	0	0000 00C0h	8000 0018h[2]	0000 00C0h	8000 0018h[3]
Select one of the above values to program in DDI_BUF_TRANS entry 9.						

[1]This row is the recommended default if a specific value is not selected.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

The following values are customized for unique motherboard configurations selected through BIOS VBT.

Customized buffer translation programming for DisplayPort											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level[1]**	**Pre-emphasis Level[1]**	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**Custom 1 (Short ULT Traces)**		**Custom 2 (Long ULX Traces)**		**Custom 3 (KBL S Short Traces)**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	400	400	0	0000 00A2h	0000 1017h	0000 00A0h	**8**000 2016h[3]	0000 00A0h	0000 1017h
1	0	1	400	600	3.5	0000 0088h	0000 4013h	0000 0088h	**8**000 5012h[3]	0000 009Bh	0000 4013h
2	0	2	400	800	6	0000 00CDh	**8**000 6011h[2]	0000 00CDh	**8**000 7011h[3]	0000 0088h	0000 6011h

Customized buffer translation programming for DisplayPort											
3	0	3	400	1000	9.5	0000 00C0h	**8**000 800Fh[2]	0000 00C0h	**8**000 9010h[3]	0000 00C0h	**8**000 800Fh[2]
4	1	0	600	600	0	0000 009Dh	0000 1017h	0000 009Dh	**8**000 2016h[3]	0000 009Bh	0000 1017h
5	1	1	600	900	3.5	0000 00C0h	**8**000 4013h[2]	0000 00C0h	**8**000 5012h[3]	0000 0088h	0000 4013h
6	1	2	600	1000	6	0000 00C0h	**8**000 6011h[2]	0000 00C0h	**8**000 7011h[3]	0000 00C0h	**8**000 6011h[2]
7	2	0	800	800	0	0000 0088h	0000 1017h	0000 0088h	**8**000 2016h[3]	0000 0097h	0000 1017h
8	2	1	800	1000	3.5	0000 00C0h	**8**000 4013h[2]	0000 00C0h	**8**000 5012h[3]	0000 00C0h	**8**000 4013h[2]

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

DDI AUX Channel

DDI_AUX_CTL

DDI_AUX_DATA

DDI_AUX_MUTEX

AUX programming sequence

A general purpose AUX functional programming sequence is provided below.

AUX Functional Sequence

Step	Description	Register	Notes
1	Display must already be initialized.		Power well1 enabled cdclk enabled IO powered up as needed for Aux on this project
1b	Power well containing Aux logic powered up	PWR_WELL_CTL	High level power well partitioning shown in display overview diagram
2	Disable DC5 and DC6 before a DDI A AUX channel transaction is sent.	DC_STATE_EN	PG1 may disable automatically for DC5 or DC6.
Skip step 3, 4 if PSR1/SRD, PSR2 or GTC are NOT enabled. Do not skip step 3,4 if PSR1/SRD, PSR2 or GTC are enabled.			
3	Enable MUTEX without changing MUTEX status	DDI_AUX_MUTEX[31]='1'	
4	Read MUTEX status	DDI_AUX_MUTEX[30]	If MUTEX status is '1', wait for 500 us and poll for MUTEX status == '0'.
5	Program AUX data registers.	DDI_AUX_DATA_*_[0-4]	
6	Program control to configure AUX and START transaction.	DDI_AUX_CTL_*	Timeout timer value must be at least 600us. Timer values: 0: 400us 1: 600 us 2: 800 us 3: 1600 us START trigger: DDI_AUX_CTL_*[31]='1'
7	Wait for AUX transaction complete.		AUX Transaction complete interrupt if set OR when DDI_AUX_CTL_*[31:30] = '01'.
8	Check that receive data has no errors	DDI_AUX_CTL_*[25]	If set: write a '1' to clear this bit and skip reading AUX data registers.
9	Read AUX data register	DDI_AUX_DATA_*_[0-4]	Condition: Aux Channel Control Register Send/Busy bit is NOT asserted
10	Clear status flags	DDI_AUX_CTL_*[30]	Transaction done status

Step	Description	Register	Notes
Skip step 11 if PSR1/SRD, PSR2 or GTC are NOT enabled. Do not skip step 11 if PSR1/SRD, PSR2 or GTC are enabled.			
11	Release MUTEX	DDI_AUX_MUTEX[31:30]= "11" OR DDI_AUX_MUTEX[31:30]="01"	Release with (is optional) or without disabling MUTEX function.

DisplayPort Transport

There is one instance of these registers per each DDI.

DP_TP_CTL

DP_TP_STATUS

Global Time Code (GTC)

Top Level GTC

GTC_CTL

GTC_DDA_M

GTC_DDA_N

GTC_LIVE

GTC Interrupt Bit Definition

GTC_IMR

GTC_IIR

DDI Level GTC

GTC_PORT_CTL
GTC_PORT_TX_CURR
GTC_PORT_TX_PREV
GTC_PORT_MISC

GTC Target Frequency Selection

For GTC top level logic, CDCLK is taken as an input and scaled to a "target frequency" which has a period which is an exact multiple of 0.5ns. This period is also known as the accumulator increment.

Once a target frequency + accumulator increment is selected, an M and N value can be picked and fine tuned to achieve the scaling from CDCLK to target frequency.

In order for the accumulated GTC Live value to match exactly with the real passage of time, the following must be true:

The target frequency selected must be the CLOSEST possible selection to CDCLK. This corresponds to rounding the accumulator increment to the NEAREST 0.5ns increment with respect to the period of CDCLK.

Example

	Frequency	Period
Given CDCLK	337.500 MHz	2.96 ns
1st closest target frequency/increment	333.333 MHz	3.00 ns
2nd closest target frequency/increment	400.000 MHz	2.50 ns
3rd closest target frequency/increment	285.714 MHz	3.50 ns
4th closest target frequency/increment	500.000 MHz	2.00 ns

Only using the 1st closest target frequency 333.333MHz / accumulator increment of 3.00 ns will result in the GTC Live Value correctly tracking the real passage of time.

Example Calculation Flow:

1. Find the period of CDCLK -- for 337.5 MHz = 2.96 ns.
2. Round to the nearest 0.5 ns -- 3.00 ns (this is your accumulator increment value).
3. Find the “target frequency” from the rounded period value of step 2 -- 333.333 MHz is the “target frequency”.
4. Find the ratio of target frequency / CDCLK = 0.987654321.
5. Choose M and N to satisfy M / N = same ratio as step 4.

South Display Engine Registers

The South Display Engine supports Hot Plug Detection, GPIO, GMBUS, Panel Power Sequencing, and Backlight Modulation.

Terminology

Access Field	Description	Should be implemented as
R/W (Read/Write)	The value written into this register will control hardware and is the same value that will be read.	Write data is stored. Read is from the stored data. Stored value is used to control hardware.
Reserved	Unused register bit. Don't assume a value for these bits. Writes have no effect.	Write data is ignored. Read is zero.
MBZ (Must Be Zero)	Always write a zero to this register.	May be implemented as Reserved or as R/W.
PBC (Preserve Bit Contents)	Software must write the original value back to this bit. This allows new features to be added using these bits.	May be implemented as Reserved or as R/W.
Read Only	The read value is determined by hardware. Writes to this bit have no effect.	Write data is ignored. Read is from a status signal or some other internal source.
Write Only	The value written into this register will control hardware. Reads return zero.	Write data is stored. Read is zero. Stored value is used to control hardware.
R/W Clear (Read/Write Clear)	Sticky status bit. Hardware will set the bit, software can clear it with a write of 1b.	Internal hardware events set a sticky bit. Read is from the sticky bit. A write of 1b clears the sticky bit.
Double Buffered	Write when desired and the written value will take effect at the time of the double buffer update point. Reads will return the written value, which is not necessarily the value being currently used to control hardware. Some have a specific arming sequence where a write to another register is required before the update can take place. This is used to ensure atomic updates of several registers.	Two stages of registers used. Write data is stored into first stage. Read is from the first stage stored data. First stage stored value is transferred to second stage storage at the double buffer update point. Second stage stored value is used to control hardware. Arm/disarm logic may be used for some registers to control the double buffer update point.
Write/Read Status	The value written into this register will control hardware. The read value is determined by hardware.	Write data is stored. Stored value is used to control hardware. Read is from a status signal or some

Access Field	Description	Should be implemented as
		other internal source.

Shared Functions

Fuses and Straps

SFUSE_STRAP

Raw Clock

RAWCLK_FREQ must be programmed to match the raw clock frequency.

Description
RAWCLK_FREQ

Description
Raw clock frequency = 24 MHz

Interrupts and Hot Plug

Description
South Display Engine Interrupt Bit Definition **SINTERRUPT** **SHPD_PULSE_CNT** **SHPD_FILTER_CNT**
SHOTPLUG_CTL **SHOTPLUG_CTL2**

Panel Power and Backlight

Panel Power

Description
PP_DIVISOR
PP_STATUS **PP_CONTROL** **PP_ON_DELAYS** **PP_OFF_DELAYS**

Backlight

This section refers to the PCH display backlight control. For CPU display backlight control, see North Display Engine Registers.

The backlight PWM output frequency is determined by the PWM clock frequency, increment, and frequency divider.

PWM output frequency = PWM clock frequency / PWM increment / PWM frequency divider

The frequency divider minimum must be greater than or equal to the number of brightness levels required by software; typically 100 or 256.

Description
PWM clock frequency = 24 MHz PWM increment = 128 or 16, selectable by software PWM frequency divider maximum = 65,536 PWM output frequency range with 100 brightness levels and increment 128 = 3 to 1,875 Hz PWM output frequency range with 100 brightness levels and increment 16 = 23 to 15,000 Hz PWM output frequency range with 256 brightness levels and increment 128 = 3 to 732 Hz PWM output frequency range with 256 brightness levels and increment 16 = 23 to 5,859 Hz
PWM clock frequency = Raw clock frequency = 19.2, 24, or 38.4 MHz. Value is found in SFUSE_STRAP. PWM increment = 1 PWM frequency divider maximum = 2^32 PWM output frequency range with 100 brightness levels and 19.2 MHz clock = 0.0004 to 192,000 Hz

Description
PWM output frequency range with 100 brightness levels and 24 MHz clock = 0.0006 to 240,000 Hz
PWM output frequency range with 100 brightness levels and 38.4 MHz clock = 0.0009 to 384,000 Hz
PWM output frequency range with 256 brightness levels and 19.2 MHz clock = 0.0004 to 75,000 Hz
PWM output frequency range with 256 brightness levels and 24 MHz clock = 0.0006 to 93,750 Hz
PWM output frequency range with 256 brightness levels and 38.4 MHz clock = 0.0009 to 150,000 Hz

Backlight Enabling Sequence

Description
1. Set frequency and duty cycle in SBLC_PWM_CTL2 Backlight Modulation Frequency and Backlight Duty Cycle.
2. Set granularity in 0xC2000 bit 0 (0 = 16, 1 = 128).
3. Enable PWM output and set polarity in SBLC_PWM_CTL1 PWM PCH Enable and Backlight Polarity.
...
4. Change duty cycle as needed in SBLC_PWM_CTL2 Backlight Duty Cycle.

If needed, granularity, polarity, and override can be programmed earlier than shown.

Backlight Registers

Description
SBLC_PWM_CTL1
SBLC_PWM_CTL2

GMBUS and GPIO

Registers

Description
GPIO_CTL - GPIO Control
GMBUS0 - GMBUS Clock/Port Select
GMBUS1 - GMBUS Command/Status
GMBUS2 - GMBUS Status
GMBUS3 - GMBUS Data Buffer
GMBUS4 - GMBUS Interrupt Mask
GMBUS5 - GMBUS 2 Byte Index

Pin Usage

These GPIO pins allow the support of simple query and control functions such as DDC interface protocols. The GMBUS controller can be used to run the interface protocol, or the GPIO pins can be manually programmed for a "bit banging" interface.

The following tables describe the expected GPIO pin to register mapping. OEMs have the ability to remap these functions onto other pins as long as the hardware limitations are observed. The GPIO pins may also be muxed with other functions such that they are only available when the other function is not being used.

Port #	Name	Pin	Pull up/down	Description
5	DDID CTLDATA	DDPD_CTRLDATA	No (Weak pull down on reset)	DDC for port D. Digital port D present strap is set if pin is 1 at rising edge of PCH_PWROK.
	DDID CTLCLK	DDPD_CTRLCLK	No	
4	DDIB CTLDATA	DDPB_CTRLDATA	No (Weak pull down on reset)	DDC for port B. Digital port B present strap is set if pin is 1 at rising edge of PCH_PWROK.
	DDIB CTLCLK	DDPB_CTRLCLK	No	
3	DDIC CTLDATA	DDPC_CTRLDATA	No (Weak pull down	DDC for port C. Digital port C present strap is set if pin is 1 at rising edge of PCH_PWROK.

			on reset)	
	DDIC CTLCLK	DDPC_CTRLCLK	No	

GPIO Programming for I2C Bit Bashing

To drive GPIO pin low, program direction to "out" and data value to "0", along with mask bits.

To drive GPIO pin high (tristate to allow external pull up to activate), program direction to "in", along with mask bit. No need to set data value to "1".

GMBUS Controller Programming Interface

The GMBUS (Graphic Management Bus) is used to access/control devices connected to the GPIO pins.

Basic features:

1. I²C compatible.
2. Bus clock frequency of 50 KHz or 100 KHz.
3. Attaches to any of the GPIO pin pairs.
4. 7-bit or 10-bit Slave Address and 8-bit or 16-bit index.
5. Double buffered data register and a 9 bit counter support 0 byte to 256 byte transfers.
6. Supports stalls generated by the slave device pulling down the clock line (Slave Stall), or delaying the slave acknowledge response.
7. Status register indicates error conditions, data buffer busy, time out, and data complete acknowledgement.
8. Detects and reports time out conditions for a stall from a slave device, or a delayed or missing slave acknowledge.
9. Interrupts may optionally be generated.
10. Does not directly support segment pointer addressing as defined by the Enhanced Display Data Channel standard.

Segment pointer addressing as defined by the Enhanced Display Data Channel standard:

1. Use bit bashing (manual GPIO programming) to complete segment pointer write **without terminating in a stop or wait cycle**.
2. Terminate bit bashing phase with both I²C lines pulled high by tri-stating the data line before the clock line. Follow EDDC requirement for response received from slave device.
3. Initiate GMBUS cycle as required to transfer EDID following normal procedure.

Sequence for GMBUS Burst Reads Greater Than 511 Bytes

- Not available until KBLPCH-H A0, SPT-LP D1, SPT-H E1

1. X=number of bytes to read.
2. Set byte count override bit GMBUS0 bit 6.
3. Set total byte count field in GMBUS1 to 511.
4. Read X bytes.
5. Issue GMBUS stop by writing 100b to GMBUS1 Bus Cycle Select.
6. Clear byte count override bit.

Example with 526 bytes

1. X=526.
2. Set byte count override bit GMBUS0 bit 6.
3. Set total byte count field in GMBUS1 to 511.
4. Read 526 bytes.
5. Write 100b to GMBUS1 Bus Cycle Select.

   Clear byte count override bit.

Display Watermark Programming

Watermark Overview

The display watermarks are used to control the display engine memory request behavior.

Description
The default settings of the watermark configuration registers will **not** allow the display engine to operate. The watermark values must be properly calculated and programmed in order to enable a display and achieve optimum power and performance. Incorrectly programmed watermark values can result in screen corruption.

The watermarks should be calculated and programmed when any of the watermark calculation inputs change. This includes planes enabling or disabling, plane source format or size changing, etc.

Besides programming the watermark registers, there are other display configuration requirements and registers that must be programmed in order for the display to operate in a low power mode, and there are memory controller configuration requirements which are not documented here.

Watermark Calculations

The display watermarks are calculated using information from the display configuration and memory latencies. The watermarks must be calculated and programmed before enabling a plane or changing a plane configuration.

For YUV 420 Planar formats, only the Y surface watermark value is calculated and programmed. Though the UV surface watermark value is not programmed seperately, it must be calculated to make sure that the UV buffer allocation satisfies the latency requirments.

The ceiling function rounds any non-integer value up to the next greater integer. Example: ceiling[0.3]=1, ceiling[2.1]=3, ceiling[4.8]=5, ceiling[4]=4

Watermark Algorithm

1. Retrieve memory latency values
    - See the Memory Values section to find the memory latency values
    - The memory values do not change after boot, so software may cache them to avoid re-reading
2. For each enabled pipe (run each time pipe configuration changes)
    A. Calculate adjusted pipe pixel rate
        I. Adjusted pipe pixel rate = pixel rate for the screen resolution
            - If there will be dynamic switching between refresh rates, either use the fastest pixel rate, or re-calculate using the current pixel rate when the refresh rate is switched
            - If plane 90 or 270 rotation is enabled, use the rotated width and height in pixel rate calculations.
        II. If TRANS_CONF Interlaced Mode == PF-ID, adjusted pipe pixel rate = adjusted pipe pixel rate * 2
        III. If pipe scaling enabled, adjusted pipe pixel rate = adjusted pipe pixel rate * pipe down scale amount
            - See the Scaling section to find the down scale amount
    B. Program WM_LINETIME Line Time = roundup[8 * pipe horizontal total pixels / adjusted pipe pixel rate MHz]
3. For each enabled plane (run each time pipe or plane configuration changes)
    A. Calculate adjusted plane pixel rate
        I. Adjusted plane pixel rate = adjusted pipe pixel rate
        II. If plane scaling enabled, adjusted plane pixel rate = adjusted plane pixel rate * plane down scale amount
            - See the Scaling section to find the down scale amount
    B. For each valid memory latency level

Plane Bytes per pixel	Minimum Scanlines for Y Tile	
	0/180 Rotation	90/270 Rotation
1 or NV12	4	16
2, P0xx, or YUV422	4	8
4	4	4
8	4	N/A

        I. Calculate method 1
            - Method 1 = memory latency microseconds * adjusted plane pixel rate MHz * plane source bytes per pixel / 512
        II. Calculate method 2

- plane bytes per line = plane source width pixels * plane source bytes per pixel
- Calculate plane blocks per line
  - If plane memory format is Linear
    - plane blocks per line = ceiling[plane bytes per line / 512] + 1
  - Else If plane memory format is Y tile
    - plane blocks per line = ceiling[Minimum Scanlines for Y tile * plane bytes per line / 512]/Minimum Scanlines for Y tile
  - Else
    - plane blocks per line = ceiling[plane bytes per line / 512]
- Method 2 = ceiling[(memory latency microseconds * adjusted plane pixel rate MHz) / Pipe horizontal total number of pixels] * plane blocks per line

III. Calculate Y tile minimum

- Y tile minimum = Minimum Scanlines for Y tile * plane blocks per line

IV. Select the watermark result

- If plane memory format is X tile or linear
  - If (((plane source bytes per pixel * pipe horizontal total number of pixels) / 512) < 1) AND ((plane bytes per line / 512) < 1) // Special case for unrealistically small horizontal total
    - Selected Result Blocks = Method 2
  - Else If (plane buffer allocation / plane blocks per line) >=1
    - Selected Result Blocks = minimum[Method 1, Method 2]
  - Else
    - Selected Result Blocks = Method 1
- Else // Y tile
  - Selected Result Blocks = maximum[Method 2, Y tile minimum]
-

V. Convert result to blocks and lines

- Result Blocks = ceiling[Selected Result Blocks] + 1
- Result Lines = ceiling[Selected Result Blocks / plane blocks per line]

this applies to all Gen9 projects

If latency level 1 through 7 and Y tile:

Result Blocks = Result Blocks + Y tile minimum

Result Lines = Result Lines + Minimum Scanlines for Y tile

If latency level 1 through 7 and not Y tile:

- Result Blocks = Result Blocks + 1

This applies to all Gen9 projects

If Render Decompression enabled and latency level 0:

Result Blocks = Result Blocks + Y tile minimum

// For latency levels 1 through 7 ensure that the their result lines/blocks >= latency level 0 lines/blocks.

For latency levels 1 through 7

If latency level result blocks < latency level 0 blocks

latency level result blocks = latency level 0 blocks

If latency level result lines < latency level 0 lines

- latency level result lines = latency level 0 lines

VI. Compare against the maximum
- If (Result Blocks >= plane buffer allocation), maximum exceeded for this latency level
- If latency level 1 through 7
  - If (Result Lines > 31), maximum exceeded for this latency level
- For YUV 420 Planar formats, perform the above check for both Y and UV planes.

4. For transition watermark

A. Calculate transition offset
- Transition Offset Blocks = Transition minimum + Transition amount
- See Transition Watermark section for transition minimum and transition amount

B. Calculate transition Y tile minimum
- Transition Y tile minimum = 2 * memory latency level 0 Y tiled minimum

C. Select the watermark result
- If plane memory format is X tile or linear
  - Result Blocks = Memory latency level 0 Selected Result Blocks + Transition Offset Blocks
- Else // Y tile
  - Result Blocks = maximum[Memory latency level 0 Selected Result Blocks, Transition Y tile minimum] + Transition Offset Blocks

D. Convert result to blocks
- Result Blocks = ceiling[Result Blocks] + 1

This applies to all Gen9 projects

If not Y tile:

- Result Blocks = Result Blocks + 1
-

E. Compare against the maximum

- If (Result Blocks >= plane buffer allocation), maximum exceeded for transition watermark
- For YUV 420 Planar formats, perform the above check for both Y and UV planes.

5. Program watermark registers
   A. For each latency level 0 to 7
      - If memory latency for this level is invalid, or the maximum was exceeded for this level or any previous level, program PLANE_WM_<latency level> Enable = 0
        - **If watermark latency level 0 exceeds the maximum, the plane must not be enabled.**
      - Else program PLANE_WM_<latency level> Enable = 1, Lines = Result Lines, Blocks = Result Blocks
        - The latency level 0 Lines value is ignored by hardware in Gen9
   B. For transition watermark
      - If the maximum was exceeded for the transition watermark, program PLANE_WM_TRANS Enable = 0
      - Else program PLANE_WM_TRANS Enable = 1, Blocks = Transition Result Blocks
        - The transition watermark Lines value is ignored by hardware
   C. Write the plane surface base address register to trigger update of the watermarks and other plane double buffered registers. This should be done only after all plane configuration is configured to match the new watermark values.

Transition Watermark

The transition watermark is used for Isochronous Priority Control (IPC). When IPC is enabled (ARB_CTL2 Enable IPC), plane read requests are sent at high priority until filling above the transition watermark, then the requests are sent at lower priority until dropping below the level 0 watermark. The lower priority requests allow other memory clients to have better memory access. If the transition watermark is not enabled, the plane behaves as if the transition watermark was programmed to the top of the plane buffer allocation. When IPC is disabled, all plane read requests are sent at high priority.

The transition watermark is programmed as a tunable amount above the level 0 watermark. Tuning to higher values will tend to cause longer periods of high priority reads followed by longer periods of lower priority reads. Tuning to lower values will tend to cause shorter periods of high and lower priority reads. The exact behavior depends on the memory bandwidth, display bandwidth, and other memory traffic in the system.

The transition watermark has a minimum value to ensure the demote does not happen before enough data has been read to meet the level 0 watermark requirements.

Transition Minimum
14 Blocks

Scaling

A scaler (pipe or plane scaler) is down scaling when it is enabled and the scaler input size is greater than the scaler output size.

Down scaling effectively increases the pixel rate. Up scaling does not reduce the pixel rate.

For plane scaling, the scaler input size is the plane size and the output size is the scaler window size.

For pipe scaling, the scaler input size is the pipe source size and the output size is the scaler window size.

Horizontal down scale amount = maximum[1, Horizontal source size / Horizontal destination size]

Vertical down scale amount = maximum[1, Vertical source size / Vertical destination size]

Total down scale amount = Horizontal down scale amount * Vertical down scale amount

Workaround	
Context:	Watermark Calculations

Workaround when calculating watermarks

Calculate arbitrated display bandwith:

For each pipe {

For each plane, except cursor, enabled on the pipe {

Plane bandwidth MB/s = pixel rate MHz * source pixel format in bytes * plane down scale amount * pipe down scale amount

}

Pipe bandwidth MB/s = Number of planes, except cursor, enabled on the pipe * Max(bandwidth from each plane) // find plane with highest bandwidth and multiply with number of enabled planes

}

Arbitrated display bandwidth = Number of pipes enabled * Max(bandwidth from each pipe) // find pipe with highest bandwidth and multiply with number of enabled pipes

If there is any Ytile plane enabled and arbitrated display bandwidth > 20% of raw system memory bandwidth (# memory channels * memory frequency * channel width in bytes) {

Double Ytile planes minimum lines and program all watermark levels accordingly. Do not change the latency values.

Increase Xtile planes watermark latency for all levels by 15us

}

If memory is dual channel single rank, Xtile limit = 35%, else Xtile limit = 60%

If there is no Ytile plane enabled and arbitrated display bandwidth > (Xtile limit * raw system memory bandwidth) {

Increase Xtile planes watermark latency for all levels by 15uS

}

System Agent Geyserville (SAGV)

On core CPUs, SAGV dynamically adjusts the system agent voltage and clock frequencies depending on power and performance requirements. The display engine access to system memory is blocked during the adjustment time.

SAGV defaults to enabled. Software must use the GT-driver pcode mailbox to disable SAGV when the display engine is not able to tolerate the blocking time.

See the Memory Values section to find the SAGV block time.

Requirement before plane enabling or configuration change: Disable SAGV if any enabled plane will not be able to enable watermarks for memory latency >= SAGV block time, or any transcoder is interlaced. Else, enable SAGV.

If software ensures single pipe configurations always have enough data buffer allocation to tolerate SAGV, it can then simply disable SAGV anytime multiple display pipes are enabled or interlace is enabled, and re-enable SAGV when switching back to a single, non-interlaced pipe.

Sequence to Disable SAGV

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low = 0x00000000.
3. Write GT Driver Mailbox Interface = 0x80000021.
4. Poll for GT Driver Mailbox Interface Run/Busy == 0x0.
    - Timeout and fail after 150 us.
5. Read GT Driver Mailbox Data Low, if bit 0 == 0x1, continue, else go to step 2.
    - If the condition in step 5 is not satisfied after cycling through steps 2-5 for 1 ms (typically <200 us), timeout and fail.
6. Continue with plane and pipe programming.

Sequence to Enable SAGV

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low = 0x00000003.
3. Write GT Driver Mailbox Interface = 0x80000021.
4. Continue with plane and pipe programming. There is no need to wait for SAGV enabling to complete.

Examples

Example pixel rate adjustments:

Pixel rate for screen resolution is 130 MHz. No interlacing. Pipe scale 1920x1080 pipe source size to 1714x1120 scaler window size. Plane scale 1920x1080 plane size to 800x600 scaler window size.

Pipe horizontal down scale amount = maximum[1, 1920 / 1714] = 1.12

Pipe vertical down scale amount = maximum[1, 1080 / 1120] = 1 **// Max condition was hit**

Pipe total down scale amount = 1.12 * 1 = 1.12

Adjusted pipe pixel rate = 130 MHz * 1.12 = 145.6 MHz

Plane horizontal down scale amount = maximum[1, 1920 / 800] = 2.4

Plane vertical down scale amount = maximum[1, 1080 / 600] = 1.8

Plane total down scale amount = 2.4 * 1.8 = 4.32

Adjusted plane pixel rate = 145.6 MHz * 4.32 = 628.99 MHz

Example method, block, and line calculations:

Plane source 4 Bpp, Plane X tile, Plane source width 1920 pixels, Horizontal total 2200 pixels, Adjusted plane pixel rate 148.5 MHz, memory latency 7.5 us

Method 1 = 148.5 MHz * 4 Bpp * 7.5 us / 512 = 8.7 blocks

Plane bytes per line = 1920 pixels * 4 Bpp = 7680 Bytes/line

Plane blocks per lines = ceiling[7680 / 512] = 15 blocks

Method 2 = ceiling[(7.5 us * 148.5 MHz) / 2200 pixels] * 15 blocks = 15 blocks

Y tile minimum = 4 * 15 blocks = 60 blocks

Result Blocks = minimum[8.7 blocks, 15 blocks] = 8.7 blocks // X tile so does not use Y tile minimum

Result Blocks = ceiling[8.7 blocks] + 1 block = 10 blocks

Result Lines = ceiling[8.7 blocks / 15] = 1 lines

Memory Values

Retrieve Memory Latency Data

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data0=0x0000_0000 (first set of latency values) and GT Driver Mailbox Data1=0x0000_0000
3. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
4. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
5. Read GT Driver Mailbox Data0 for the first set of memory latency values
6. Write GT Driver Mailbox Data0=0x0000_0001 (second set of latency values) and GT Driver Mailbox Data1=0x0000_0000
7. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
8. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
9. Read GT Driver Mailbox Data0 for the second set of memory latency values

Memory Latency Data Definition

First Set		
Data0 Bit	**Name**	**Description**
31:24	Level 3	Number of microseconds for level 3.
23:16	Level 2	Number of microseconds for level 2.
15:8	Level 1	Number of microseconds for level 1.
7:0	Level 0	Number of microseconds for level 0.

Second Set		
Data0 Bit	**Name**	**Description**
31:24	Level 7	Number of microseconds for level 7.
23:16	Level 6	Number of microseconds for level 6.
15:8	Level 5	Number of microseconds for level 5.
7:0	Level 4	Number of microseconds for level 4.

If level 1 or any higher level has a value of 0x00, that level and any higher levels are unused and invalid, so the associated watermark registers must not be enabled.

It is allowed to have the same value in adjacent levels.

Workaround	
Context:	Display Watermark Programming
The mailbox response data may not account for memory read latency. If the mailbox response data for level 0 is 0us, add 2 microseconds to the result for each valid level.	

SAGV Block Time

SAGV Block Time
30 us

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 2d: Command Reference: Structures

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

3DSTATE_AMFS_POINTERS_REMOVED 1
3DSTATE_CONSTANT_ALL_DATA 2
3DSTATE_CONSTANT(Body) 3
A32 Buffer Base Address Message Header Control 6
A32 Scaled Header Present Message Descriptor Control Field 7
A32 Sideband Scale and Offset Enable Message Descriptor Control Field 8
A64 Data Size Message Descriptor Control Field 9
A64 Dual Oword Block Message Header 10
A64 Hword Block Message Header 11
A64 Hword Data Blocks Message Descriptor Control Field 12
A64 Oword Block Message Header 13
A64 Oword Data Blocks Message Descriptor Control Field 14
A64 Oword Dual Data Blocks Message Descriptor Control Field 15
A64 Scaled Data Port 2 Message Header 16
A64 Scaled Header Present Message Descriptor Control Field 17
A64 Sideband Scale and Offset Enable Message Descriptor Control Field 18
AddrSubRegNum 19
AMFS_STATE_REMOVED 20
Any Binding Table Index Message Descriptor Control Field 21
Atomic Float Binary Operation Message Descriptor Control Field 22
Atomic Float Trinary Operation Message Descriptor Control Field 23
Atomic Integer Binary Operation Message Descriptor Control Field 24
Atomic Integer Trinary Operation Message Descriptor Control Field 25
Atomic Integer Unary Operation Message Descriptor Control Field 26
Audio Power State Format 27
AVC CABAC 28
AVC CAVLC 29
AVS_Inline_DMEM 31
BaseAddress4KByteAligned 34
BCS Hardware-Detected Error Bit Definitions 35
BINDING_TABLE_EDIT_ENTRY 36

BINDING_TABLE_STATE 37
Bit Definition for Interrupt Control Registers - Blitter 38
Bit Definition for Interrupt Control Registers - Media#1 40
Bit Definition for Interrupt Control Registers - Media#2 42
Bit Definition for Interrupt Control Registers - Render 44
Bit Definition for Interrupt Control Registers - Video Enhancement 46
BLEND_STATE 48
BLEND_STATE_ENTRY 50
Block Dimensions Message Header Control 55
Block Message Header 56
BR00 - BLT Opcode and Control 57
BR01 - Setup BLT Raster OP, Control, and Destination Offset 61
BR05 - Setup Expansion Background Color 64
BR06 - Setup Expansion Foreground Color 65
BR07 - Setup Blit Color Pattern Address Lower Order Address bits 66
BR09 - Destination Address Lower Order Address Bits 67
BR11 - BLT Source Pitch (Offset) 68
BR12 - Source Address Lower order Address bits 69
BR13 - BLT Raster OP, Control, and Destination Pitch 70
BR14 - Destination Width and Height 73
BR15 - Color Pattern Address Lower order Address bits 74
BR16 - Pattern Expansion Background and Solid Pattern Color 75
BR17 - Pattern Expansion Foreground Color 76
BR18 - Source Expansion Background and Destination Color 77
BR19 - Source Expansion Foreground Color 78
BR27 - Destination Higher Order Address 79
BR28 - Source Higher Order Address 80
BR29 - Color Pattern Higher Order Address 81
BR30 - Setup Blit Color Pattern Higher Order Address 82
Byte Masked Media Block Message Header 83
Byte Masked Media Block Message Header Control 84
CC_VIEWPORT 86
Channel Mask Message Descriptor Control Field 87

Channel Mode Message Descriptor Control Field 88
Clock Gating Disable Format 89
Clock Gating Disable Format 90
COLOR_CALC_STATE 91
COLOR_PROCESSING_STATE - ACE State 93
COLOR_PROCESSING_STATE - CSC State 99
COLOR_PROCESSING_STATE - PROCAMP State 103
COLOR_PROCESSING_STATE - STD/STE State 104
COLOR_PROCESSING_STATE - TCC State 118
Color Calculator State Pointer Message Header Control 123
Color Code Message Header Control 124
Context Descriptor Format 125
Context Status 129
CSC COEFFICIENT FORMAT 132
Data Port 0 Message Types 133
Data Port 1 Message Types 134
Data Port 2 Extended Message Descriptor 136
Data Port 2 Message Types 137
Data Port Bindless Surface Extended Message Descriptor 138
Data Size Message Descriptor Control Field 139
Deptrh Clear Value Format 140
Display Engine Render Response Message Definition 141
DstRegNum 143
DstSubRegNum 144
Dword Data Payload Register 145
Dword SIMD4x2 Atomic CMPWR Message Data Payload 146
Dword SIMD4x2 Atomic Operation Message Data Payload 147
Dword SIMD4x2 Data Payload 148
Dword SIMD8 Atomic Operation CMPWR Message Data Payload 149
Dword SIMD8 Data Payload 150
Dword SIMD16 Atomic Operation CMPWR Message Data Payload 151
Dword SIMD16 Data Payload 152
DX9_CONSTANTB_ENTRY 153

DX9_CONSTANTF_ENTRY 154
DX9_CONSTANTI_ENTRY 155
Encoder Control State Parameters0 156
Encoder Statistics Format 162
EU_INSTRUCTION_BASIC_ONE_SRC 166
EU_INSTRUCTION_BASIC_THREE_SRC 167
EU_INSTRUCTION_BASIC_TWO_SRC 170
EU_INSTRUCTION_BRANCH_CONDITIONAL 171
EU_INSTRUCTION_BRANCH_ONE_SRC 173
EU_INSTRUCTION_BRANCH_TWO_SRC 174
EU_INSTRUCTION_COMPACT_THREE_SRC 175
EU_INSTRUCTION_COMPACT_TWO_SRC 178
EU_INSTRUCTION_CONTROLS_A 184
EU_INSTRUCTION_CONTROLS_B 187
EU_INSTRUCTION_CONTROLS 189
EU_INSTRUCTION_HEADER 190
EU_INSTRUCTION_ILLEGAL 191
EU_INSTRUCTION_MATH 192
EU_INSTRUCTION_NOP 193
EU_INSTRUCTION_OPERAND_CONTROLS 194
EU_INSTRUCTION_OPERAND_DST_ALIGN1 196
EU_INSTRUCTION_OPERAND_DST_ALIGN16 197
EU_INSTRUCTION_OPERAND_SEND_MSG 198
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 199
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 200
EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC 201
EU_INSTRUCTION_SEND 202
EU_INSTRUCTION_SENDS 204
EU_INSTRUCTION_SOURCES_IMM32 206
EU_INSTRUCTION_SOURCES_REG 207
EU_INSTRUCTION_SOURCES_REG_IMM 208
EU_INSTRUCTION_SOURCES_REG_REG 209
Execution_Unit_Extended_Message_Descriptor 211

Extended Message Descriptor - Execution Unit 212
Extended Message Descriptor Render Target 213
Extended Message Descriptor - Sampling Engine 214
ExtMsgDescpt 215
ExtMsgDescptImmediate 217
FFTID Message Header Control 219
Filter_Coefficient 220
Filter_Coefficients 221
FrameDeltaQp 222
FrameDeltaQpRange 223
FunctionControl 224
GATHER_CONSTANT_ENTRY 225
GraphicsAddress63-0 226
GraphicsAddress63-1 227
GraphicsAddress63-2 228
GraphicsAddress63-3 229
GraphicsAddress63-4 230
GraphicsAddress63-5 231
GraphicsAddress63-6 232
GraphicsAddress63-12 233
GTC Interrupt Bit Definition 234
Half Precision Dual Source SIMD8 Message Data Payload Register 235
Half Precision OM Replicated SIMD16 Render Target Data Payload 237
Half Precision OM S0A SIMD8 Render Target Data Payload 238
Half Precision OM S0A SIMD16 Render Target Data Payload 239
Half Precision OM SIMD8 Dual Source Render Target Data Payload 240
Half Precision OM SIMD8 Render Target Data Payload 241
Half Precision OM SIMD16 Render Target Data Payload 242
Half Precision OS OM S0A SIMD8 Render Target Data Payload 243
Half Precision OS OM SIMD8 Dual Source Render Target Data Payload 245
Half Precision OS OM SIMD8 Render Target Data Payload 246
Half Precision OS S0A SIMD8 Render Target Data Payload 247
Half Precision OS SIMD8 Dual Source Render Target Data Payload 248

Half Precision OS SIMD8 Render Target Data Payload 249
Half Precision OS SZ OM S0A SIMD8 Render Target Data Payload 250
Half Precision OS SZ OM SIMD8 Dual Source Render Target Data Payload 252
Half Precision OS SZ OM SIMD8 Render Target Data Payload 253
Half Precision OS SZ S0A SIMD8 Render Target Data Payload 254
Half Precision OS SZ SIMD8 Dual Source Render Target Data Payload 255
Half Precision OS SZ SIMD8 Render Target Data Payload 256
Half Precision Replicated Pixel Render Target Data Payload Register 257
Half Precision Replicated SIMD16 Render Target Data Payload 258
Half Precision S0A SIMD8 Render Target Data Payload 259
Half Precision S0A SIMD16 Render Target Data Payload 260
Half Precision SIMD8 Dual Source Render Target Data Payload 261
Half Precision SIMD8 Message Data Payload Register 262
Half Precision SIMD8 Render Target Data Payload 263
Half Precision SIMD16 Message Data Payload Register 264
Half Precision SIMD16 Render Target Data Payload 266
Half Precision SZ OM S0A SIMD8 Render Target Data Payload 267
Half Precision SZ OM S0A SIMD16 Render Target Data Payload 269
Half Precision SZ OM SIMD8 Dual Source Render Target Data Payload 271
Half Precision SZ OM SIMD8 Render Target Data Payload 272
Half Precision SZ OM SIMD16 Render Target Data Payload 273
Half Precision SZ S0A SIMD8 Render Target Data Payload 275
Half Precision SZ S0A SIMD16 Render Target Data Payload 276
Half Precision SZ SIMD8 Dual Source Render Target Data Payload 278
Half Precision SZ SIMD8 Render Target Data Payload 279
Half Precision SZ SIMD16 Render Target Data Payload 280
Hardware-Detected Error Bit Definitions 281
Hardware Status Page Layout 282
HCP_PAK_INSERT_OBJECT_INDIRECT_PAYLOAD 287
Header Forbidden Message Descriptor Control Field 288
Header Present Message Descriptor Control Field 289
Header Required Message Descriptor Control Field 290
HEVC_ARBITRATION_PRIORITY 291

HW Generated BINDING_TABLE_STATE 292
Hword 1 Block Data Payload 293
Hword 2 Block Data Payload 294
Hword 4 Block Data Payload 295
Hword 8 Block Data Payload 296
Hword Channel Mode Message Header Control 298
Hword Register Blocks Message Descriptor Control Field 299
Ignored Message Header 300
Inline Data Description for MFD_AVC_BSD_Object 301
Inline Data Description - VP8 PAK OBJECT 309
INTERFACE_DESCRIPTOR_DATA 315
INTERRUPT 321
Invalidate After Read Message Descriptor Control Field 323
JPEG 324
LOD Message Address Payload Control 325
Lower Oword Block Data Payload 326
LRI Data Entry 327
MEDIA_SURFACE_STATE 328
MEMORY_OBJECT_CONTROL_STATE 338
MemoryAddressAttributes 339
Merged Media Block Message Header 341
Merged Media Block Message Header Control 342
Message Descriptor - Render Target Write 345
Message Descriptor - Sampling Engine 347
MFD_MPEG2_BSD_OBJECT Inline Data Description 349
MPEG2 351
MPEG4-2_Inline_DMEM 352
MSAA Sample Number Message Address Control 358
MsgDescpt31 359
Normal Media Block Message Header 360
Normal Media Block Message Header Control 361
oMask Message Data Payload Register 363
OM Replicated SIMD16 Render Target Data Payload 365

OM S0A SIMD8 Render Target Data Payload 366
OM S0A SIMD16 Render Target Data Payload 367
OM SIMD8 Dual Source Render Target Data Payload 369
OM SIMD8 Render Target Data Payload 371
OM SIMD16 Render Target Data Payload 372
OS OM S0A SIMD8 Render Target Data Payload 374
OS OM SIMD8 Dual Source Render Target Data Payload 376
OS OM SIMD8 Render Target Data Payload 378
OS S0A SIMD8 Render Target Data Payload 379
OS SIMD8 Dual Source Render Target Data Payload 380
OS SIMD8 Render Target Data Payload 382
OS SZ OM S0A SIMD8 Render Target Data Payload 383
OS SZ OM SIMD8 Dual Source Render Target Data Payload 385
OS SZ OM SIMD8 Render Target Data Payload 387
OS SZ S0A SIMD8 Render Target Data Payload 389
OS SZ SIMD8 Dual Source Render Target Data Payload 391
OS SZ SIMD8 Render Target Data Payload 393
Oword 1 Dual Block Data Payload 394
Oword 2 Block Data Payload 395
Oword 4 Block Data Payload 396
Oword 4 Dual Block Data Payload 397
Oword 8 Block Data Payload 399
Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload 400
Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload 401
Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload 402
Oword Data Blocks Message Descriptor Control Field 404
Oword Data Payload Register 405
Oword Dual Data Blocks Message Descriptor Control Field 406
PALETTE_ENTRY 407
Performance Counter Report Format 101b 408
Per Thread Scratch Space Message Header Control 411
PIXEL_HASH_TABLE_1BIT_32ENTRY 412
PIXEL_HASH_TABLE_1BIT_64ENTRY 413

PIXEL_HASH_TABLE_1BIT_128ENTRY 415
PIXEL_HASH_TABLE_2BIT_64ENTRY 417
PIXEL_HASH_TABLE_2BIT_128ENTRY 424
Pixel Masked Media Block Message Header 438
Pixel Masked Media Block Message Header Control 439
Pixel Sample Mask Message Header Control 441
Pixel Sample Mask Render Target Message Header Control 442
Power Clock State Format 443
PPHWSP_LAYOUT 445
Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload 450
Qword Data Payload Register 452
Qword SIMD4x2 Atomic CMPWR8B Message Data Payload 453
Qword SIMD4x2 Atomic Operation Message Data Payload 454
Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload 455
Qword SIMD8 Atomic Operation CMPWR Message Data Payload 456
Qword SIMD8 Atomic Operation Return Data Message Data Payload 457
Qword SIMD8 Data Payload 458
Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload 459
Qword SIMD16 Atomic Operation Return Data Message Data Payload 461
Qword SIMD16 Data Payload 462
Read-Only Data Port Message Types 463
Read Surface Info 32-Bit Address Payload 464
Read Surface Info Data Payload 465
RENDER_SURFACE_STATE 467
Render Data Port Message Types 499
Render Target Index Message Header Control 500
Render Target Message Header 501
Render Target Message Header Control 503
Replicated Pixel Render Target Data Payload Register 505
Replicated SIMD16 Render Target Data Payload 506
Reversed SIMD Mode 2 Message Descriptor Control Field 507
RoundingPrecisionTable_3_Bits 508
S0A SIMD8 Render Target Data Payload 509

S0A SIMD16 Render Target Data Payload 510
SAMPLER_BORDER_COLOR_STATE 512
SAMPLER_INDIRECT_STATE_BORDER_COLOR 514
SAMPLER_INDIRECT_STATE 517
SAMPLER_STATE_8x8_AVS_COEFFICIENTS 520
SAMPLER_STATE_8x8_AVS 524
SAMPLER_STATE 538
SCISSOR_RECT 550
Scratch Hword Block Message Header 551
SF_CLIP_VIEWPORT 552
SF_OUTPUT_ATTRIBUTE_DETAIL 554
SFC_8x8_AVS_COEFFICIENTS 556
SIMD4x2 MSAA Typed Surface 32-Bit Address Payload 559
SIMD4x2 Typed Surface 32-Bit Address Payload 561
SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload 562
SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload 563
SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload 564
SIMD4x2 32-Bit Address Payload 565
SIMD8 Dual Source Render Target Data Payload 566
SIMD8 LOD Message Address Payload Control 568
SIMD8 MSAA Typed Surface 32-Bit Address Payload 569
SIMD8 Render Target Data Payload 571
SIMD8 Typed Surface 32-Bit Address Payload 572
SIMD8 Untyped BUFFER Surface 32-Bit Address Payload 573
SIMD8 Untyped BUFFER Surface 64-Bit Address Payload 574
SIMD8 Untyped STRBUF Surface 32-Bit Address Payload 575
SIMD16 Render Target Data Payload 576
SIMD16 Untyped BUFFER Surface 32-Bit Address Payload 578
SIMD16 Untyped BUFFER Surface 64-Bit Address Payload 579
SIMD16 Untyped STRBUF Surface 32-Bit Address Payload 580
SIMD 32-Bit Address Payload Control 581
SIMD 64-Bit Address Payload Control 582
SIMD8 32-Bit Address Payload 583

SIMD8 64-Bit Address Payload 584
SIMD16 32-Bit Address Payload 585
SIMD16 64-Bit Address Payload 586
SIMD Mode 2 Message Descriptor Control Field 587
SIMD Mode 3 Message Descriptor Control Field 588
SLM Surface Pixel Mask Message Header 589
Slot Group 2 Message Descriptor Control Field 590
Slot Group 3 Message Descriptor Control Field 591
Slot Group Select Render Cache Message Descriptor Control Field 592
SO_DECL 593
SO_DECL_ENTRY 595
Split_coding_unit_flags 596
SplitBaseAddress4KByteAligned 597
SplitBaseAddress64ByteAligned 598
SrcRegNum 599
SrcSubRegNum 600
SRD Interrupt Bit Definition 601
Stateless Binding Table Index Message Descriptor Control Field 603
Stateless Block Message Header 604
Stateless Scaled Data Port 2 Message Header 605
Stateless Scaled Data Port 2 Pixel Mask Message Header 606
Stateless Surface Message Header 607
Stateless Surface Pixel Mask Message Header 608
Stencil Message Data Payload Register 609
Subset Atomic Integer Trinary Operation Message Descriptor Control Field 611
Subset Reversed SIMD Mode 2 Message Descriptor Control Field 612
Subset SIMD Mode 2 Message Descriptor Control Field 613
Subset SIMD Mode 3 Message Descriptor Control Field 614
Subspan Render Target Message Header Control 615
Surface Binding Table Index Message Descriptor Control Field 616
Surface or Stateless Binding Table Index Message Descriptor Control Field 617
Surface Pitch Message Header Control 618
Surface Pixel Mask Message Header 619

SW Generated BINDING_TABLE_STATE 620
SZ OM S0A SIMD8 Render Target Data Payload 621
SZ OM S0A SIMD16 Render Target Data Payload 623
SZ OM SIMD8 Dual Source Render Target Data Payload 625
SZ OM SIMD8 Render Target Data Payload 627
SZ OM SIMD16 Render Target Data Payload 628
SZ S0A SIMD8 Render Target Data Payload 630
SZ S0A SIMD16 Render Target Data Payload 631
SZ SIMD8 Dual Source Render Target Data Payload 633
SZ SIMD8 Render Target Data Payload 635
SZ SIMD16 Render Target Data Payload 636
Thread EOT Message Descriptor 638
Thread Spawn Message Descriptor 639
TileW SIMD8 Data Control Dword 641
TileW SIMD8 Data Payload 642
Transpose Message Header 643
Untyped Write Channel Mask Message Descriptor Control Field 644
Upper Oword Block Data Payload 645
VC1 646
VCS Hardware-Detected Error Bit Definitions 647
VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS 648
VEBOX_ACE_LACE_STATE 649
VEBOX_ALPHA_AOI_STATE 656
VEBOX_CAPTURE_PIPE_STATE 658
VEBOX_CCM_STATE 662
VEBOX_Ch_Dir_Filter_Coefficient 665
VEBOX_CSC_STATE 666
VEBOX_DNDI_STATE 670
VEBOX_Filter_Coefficient 680
VEBOX_FORWARD_GAMMA_CORRECTION_STATE 681
VEBOX_FRONT_END_CSC_STATE 682
VEBOX_GAMUT_STATE 685
VEBOX_IECP_STATE 702

VEBOX_PROCAMP_STATE 704
VEBOX_RGB_TO_GAMMA_CORRECTION 705
VEBOX_STD_STE_STATE 706
VEBOX_TCC_STATE 721
VEBOX_VERTEX_TABLE 726
VECS Hardware-Detected Error Bit Definitions 729
VERTEX_BUFFER_STATE 730
VERTEX_ELEMENT_STATE 732
Vertical Line Stride Override Message Descriptor Control Field 736
VFE_STATE_EX 737
VP8 Encoder StreamOut Format 740

3DSTATE_AMFS_POINTERS_REMOVED

3DSTATE_AMFS_POINTERS_REMOVED		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Reserved** Format: MBZ

3DSTATE_CONSTANT_ALL_DATA

3DSTATE_CONSTANT_ALL_DATA		
Source:	RenderCS	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:5	**Pointer To Constant Buffer** Format: GraphicsAddress63-5 The value of this field is the virtual address of the location of the push constant buffer. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Constant Buffer Read Length** Format: U5 This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of the read length fields for all pointers must be less than or equal to the size of 64 • Zero means there no data to fetch for this buffer pointer.

3DSTATE_CONSTANT(Body)

3DSTATE_CONSTANT(Body)		
Source:	RenderCS	
Size (in bits):	320	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:16	**Constant Buffer 1 Read Length** Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 1. • If disabled, the **Pointer to Constant Buffer 1** must be programmed to zero. if gather constant are enabled, this field must be non-zero if a there was a preceding corresponding 3DSTATE_GATHER_CONSTANT_*., otherwise this field must be zero.
	15:0	**Constant Buffer 0 Read Length** Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 0. • If disabled, the **Pointer to Constant Buffer 0** must be programmed to zero.
1	31:16	**Constant Buffer 3 Read Length** Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 3. • If disabled, the **Pointer to Constant Buffer 3** must be programmed to zero.
	15:0	**Constant Buffer 2 Read Length** Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes**

3DSTATE_CONSTANT(Body)

		• The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 2. • If disabled, the **Pointer to Constant Buffer 2** must be programmed to zero.
2..3	63:5	**Pointer To Constant Buffer 0** Format: GraphicsAddress63-5 **Description** When CONSTANT_BUFFER Address Offset Disable in CS_DEBUG_MODE2 register is set, the value of this field is the virtual address of the location of the push constant buffer. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. When CONSTANT_BUFFER Address Offset Disable in CS_DEBUG_MODE2 register is cleared, the value of this field is the offset into the Dynamic State Base Address. Only [47:5] of the field are added to the base address to generate the virtual address to be fetched from memory.
	4:0	**Reserved** Format: MBZ
4..5	63:5	**Pointer To Constant Buffer 1** Format: GraphicsAddress63-5 This field points to the location of Constant Buffer 1. If gather constants are enabled This field is an offset of constant Buffer1 from the Gather Pool BASE ADDRESS. If gather constants is disabled, the value of this field is the virtual address of the location of the push constant buffer. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Reserved** Format: MBZ
6..7	63:5	**Pointer To Constant Buffer 2** Format: GraphicsAddress63-5 The value of this field is the virtual address of the location of the push constant buffer 2. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.

3DSTATE_CONSTANT(Body)		
	4:0	**Reserved** Format: MBZ
8..9	63:5	**Pointer To Constant Buffer 3** Format: GraphicsAddress63-5 The value of this field is the virtual address of the location of the push constant buffer 3. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Reserved** Format: MBZ

A32 Buffer Base Address Message Header Control

MHC_A32_BBA - A32 Buffer Base Address Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:10	**Buffer Base Address Offset** Format: GeneralStateOffset[31:10] Specifies the base address offset page [31:10] for A32 stateless messages. **Restriction** Restriction : When using stateless A32 Data Port messages, General State Base Address[47:12] + Buffer Base Address[31:10] must be less than 2^48. It is illegal for this to be greater or equal than 2^48.
	9:0	**Reserved** Format: MBZ Ignored.

A32 Scaled Header Present Message Descriptor Control Field

MDC_A32_MHP - A32 Scaled Header Present Message Descriptor Control Field

Source:	BSpec
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Format: Enumeration Specifies if the message uses the optional message header to modify the access type and address calculation, in combination with the MDC_A32_SSO field.

Value	Name	Description
0h	No	Message header is not present. The Sideband Scale Offset field from the Message Descriptor are used as offsets with the Address Payload.
1h	Yes	Message header is present. It specifies a Base Address Offset and a Scale pitch to use with each of the offsets from the Address Payload.

Programming Notes
The access is an SLM access if the Sideband Scale Offset is enabled when the Message Header is not present; otherwise the access is a A32 access.

A32 Sideband Scale and Offset Enable Message Descriptor Control Field

MDC_A32_SBSO - A32 Sideband Scale and Offset Enable Message Descriptor Control Field

Source: BSpec
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7	**Sideband Offset Enable** Format: Enable When set, enables adding the 16-bit offset from the Sideband to all the offsets in the Address Payload for the A32 or SLM access. The 16-bit Sideband Offset is specified in the extended function control field in the SEND instruction. **Programming Notes** If set and a Message Header is not present, then the access is an SLM access and the Sideband Offset and Scale are used as offsets with the Address Payload. Otherwise, the access is an A32 access, and either or both of the Message Header and Sideband Offsets are added to the offsets in the Address Payload.
	6:0	**Scale** Format: U7 Specifies the scale pitch to be used for SLM messages as (#bytes-1). **Programming Notes** If this is a A32 message, this field is ignored and the scale pitch is supplied by the Message Header. If this is an A32 message and no Message Header present, then the scale is 0 (i.e. pitch = 1 byte).

A64 Data Size Message Descriptor Control Field

MDC_A64_DS - A64 Data Size Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**Data Size** Format: Enumeration Specifies the number of data elements to be read or written

Value	Name	Description	Programming Notes
00h	DE1	1 data element (B, DW, QW)	
01h	DE2	2 data elements (B, DW, QW)	
02h	DE4	4 data elements (B, DW, QW)	
03h	DE8	8 data elements (B, DW, QW)	Restriction : This setting is supported for DW and QW but not for B. For bytes, the maximum number of data elements is 4.

Restriction
Restriction : The number of elements is constrained by SIMD Mode and Data Width. The max data payload limit is 256B: 2 elements SIMD16 QW, 4 elements SIMD16 DW, or 4 elements SIMD8 QW.

A64 Dual Oword Block Message Header

MH_A64_OWDB - A64 Dual Oword Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**BlockOffset0** Format: U64 Specifies the U64 byte offset of Oword Block 0. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** Restriction : The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2..3	63:0	**BlockOffset1** Format: U64 Specifies the U64 byte offset of Oword Block 1. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** Restriction : The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
4..7	127:0	**Reserved** Format: Ignore Ignored

A64 Hword Block Message Header

MH_A64_HWB - A64 Hword Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**BlockOffset** Format: U64 Specifies the U64 byte offset of Oword block. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** Restriction : The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2..4	95:0	**Reserved** Format: Ignore Ignored
5	31:0	**Hword Channel Mode** Format: MHC_A64_CMODE Specifies the Hword Channel Mode
6..7	63:0	**Reserved** Format: Ignore Ignored

A64 Hword Data Blocks Message Descriptor Control Field

MDC_A64_DB_HW - A64 Hword Data Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 3
Default Value: 0x00000001

DWord	Bit	Description
0	2:0	**Data Blocks** Format: Enumeration Specifies the number of Hwords to be read or written

Value	Name	Description
01h	HW1 **[Default]**	1 Hword block
02h	HW2	2 Hword blocks
03h	HW4	4 Hword blocks
04h	HW8	8 Hword blocks
Others	Reserved	Ignored

A64 Oword Block Message Header

MH_A64_OWB - A64 Oword Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**BlockOffset** Format: U64 Specifies the U64 byte offset of Oword block. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** Restriction : The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2..7	191:0	**Reserved** Format: Ignore Ignored

A64 Oword Data Blocks Message Descriptor Control Field

MDC_A64_DB_OW - A64 Oword Data Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2:0	**Data Blocks** Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description
00h	OW1L	1 Oword, read into or written from the low 128 bits of the destination register
01h	OW1U	1 Oword, read into or written from the high 128 bits of the destination register
02h	OW2	2 Owords
03h	OW4	4 Owords
04h	OW8	8 Owords
Others	Reserved	Ignored

A64 Oword Dual Data Blocks Message Descriptor Control Field

MDC_A64_DB_OWD - A64 Oword Dual Data Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 3
Default Value: 0x00000001

DWord	Bit	Description
0	2:0	**Data Blocks** Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description
01h	OWD1 **[Default]**	1 Hword register, 2 Owords
03h	OWD4	4 Hword registers, 8 Owords
Others	Reserved	Ignored

A64 Scaled Data Port 2 Message Header

MH2_A64 - A64 Scaled Data Port 2 Message Header

Source: DataPort 2

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Surface Pitch** Format: MHC_PITCH Specifies the 16-bit surface pitch to use with scaled A64 stateless messages.
1..3	95:0	**Reserved** Format: Ignore Ignored
4..5	63:0	**Buffer Base Address** Format: GraphicsAddress[63:0] Specifies the surface base address[63:0] for scaled A64 stateless messages. The address must be Dword aligned (bits [1:0] = 0). **Programming Notes** If the Buffer Base Address is not in 48-bit canonical form, the access is Out-of-Bounds. Stateless A64 bounds checking rules specify that any address calculation that crosses the canonical address boundary is treated as out-of-bounds. A64 address calculations always use positive offsets, and do not support programming language constructs with negative offsets or indices (e.g. array[-1]).
6..7	63:0	**Reserved** Format: Ignore Ignored

A64 Scaled Header Present Message Descriptor Control Field

MDC_A64_MHP - A64 Scaled Header Present Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Format: Enumeration Specifies if the message uses the optional message header to modify the A64 address calculation, in combination with MDC_A64_SSO field. **Value / Name / Description** 0h / No / Message header is not present 1h / Yes / Message header is present **Programming Notes** The access is Out-of-Bounds if the SideBand Offset is enabled when the Message Header is not present.

A64 Sideband Scale and Offset Enable Message Descriptor Control Field

MDC_A64_SBSO - A64 Sideband Scale and Offset Enable Message Descriptor Control Field

Source: BSpec
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7	**Sideband Offset Enable** Format: Enable If set, the Sideband Offset is added to the Message Header's Base Address Offset. The 16-bit Sideband Offset is specified in the extended function control field in the SEND instruction. **Programming Notes** It is an illegal A64 encoding to enable the SideBand Offset when the Message Header is not present. That encoding combination is an SLM access in A32 message, but in an A64 message the illegal access is handled as Out-of-Bounds.
	6:0	**Reserved** Format: MBZ Ignored

AddrSubRegNum

AddrSubRegNum		
Source:	EuIsa	
Size (in bits):	4	
Default Value:	0x00000000	

Address Subregister Number
This field provides the subregister number for the address register. The address register contains 8 sub-registers. The size of each subregister is one word. The address register contains the register address of the operand, when the operand is in register-indirect addressing mode.
This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand.
This field is present if the operand is in register-indirect addressing mode; it is not present if the operand is directly addressed.
An address subregister used for indirect addressing is often called an index register.

DWord	Bit	Description
0	3:0	**Address Subregister Number**

Value	Name
0-15	Address Subregister Number

AMFS_STATE_REMOVED

AMFS_STATE_REMOVED		
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

Any Binding Table Index Message Descriptor Control Field

MDC_BTS_SLM_A32 - Any Binding Table Index Message Descriptor Control Field

Source: BSpec
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Format: Enumeration Specifies the surface for the message, which can be Surface State Model, SLM or Stateless.

Value	Name	Description
00h-0EFh	BTS	Index of Binding Table State Surfaces
F0h-0FBh	Reserved	Reserved for future use
0FCh	SSO	Specifies a Surface State Offset supplied by the extended message descriptor
0FEh	SLM	Specifies an SLM access
0FFh	A32_A64	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).

Restriction
Restriction : When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Atomic Float Binary Operation Message Descriptor Control Field

MDC_FOP2 - Atomic Float Binary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000001

DWord	Bit	Description
0	1:0	**Atomic Float Operation Type** Format: Enumeration Specifies the atomic float binary operation to be performed

Value	Name	Description	Programming Notes
01h	AOP_FMAX **[Default]**	new_dst = fmax(old_dst, src0)	The fmax operation implements the IEEE specification, which differs slightly from the DX specifications when a source operand is a sNaN. fmax(x,qNaN) = fmax(qNaN,x) = x fmax(x,sNaN) = fmax(sNaN,x) = quietize(sNaN) fmax(sNaN,sNaN) = fmax(sNaN,qNaN) = fmax(qNan,sNaN) = quietize(sNaN) fmax(qNaN,qNan) = qNaN Fmax with sNaN operand returns sNaN instead of quietize(sNaN) Fmax(-0,+0) = -0. Should be +0, to match EU Fmax instruction.
02h	AOP_FMIN	new_dst = fmin(old_dst, src0)	The fmin operation implements the IEEE specification, which differs slightly from the DX specifications when a source operand is a sNaN. fmin(x,qNaN) = fmin(qNaN,x) = x fmin(x,sNaN) = fmin(sNaN,x) = quietize(sNaN) fmin(sNaN,sNaN) = fmin(sNaN,qNaN) = fmin(qNan,sNaN) = quietize(sNaN) fmin(qNaN,qNan) = qNaN Fmin with sNaN operand returns sNaN instead of quietize(sNaN) Fmin(+0,-0) = +0. Should be -0, to match EU Fmin instruction.
Others	Reserved	Ignored	

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Float Trinary Operation Message Descriptor Control Field

MDC_FOP3 - Atomic Float Trinary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000003

DWord	Bit	Description
0	1:0	**Atomic Float Operation Type** Format: Enumeration Specifies the atomic float trinary operation to be performed

Value	Name	Description	Programming Notes
03h	AOP_FCMPWR **[Default]**	new_dst = (src0 == old_dst) ? src1 : old_dst	The fcmpwr operation performs the comparison using IEEE specification rules, and performs the store as a raw move (so SNaN is not quietized). fcmpwr(NaN,x,y) = NaN fcmpwr(x, NaN,y) = x fcmpwr(x,x, NaN) = NaN
Others	Reserved	Ignored	

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Integer Binary Operation Message Descriptor Control Field

MDC_AOP2 - Atomic Integer Binary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x00000001

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Format: Enumeration Specifies the atomic integer binary operation to be performed

Value	Name	Description
01h	AOP_AND **[Default]**	new_dst = old_dst AND src0
02h	AOP_OR	new_dst = old_dst | src0
03h	AOP_XOR	new_dst = old_dst ^ src0
04h	AOP_MOV	new_dst = src0
07h	AOP_ADD	new_dst = old_dst + src0
08h	AOP_SUB	new_dst = old_dst - src0
09h	AOP_REVSUB	new_dst = src0 - old_dst
0Ah	AOP_IMAX	new_dst = imax(old_dst, src0)
0Bh	AOP_IMIN	new_dst = imin(old_dst, src0)
0Ch	AOP_UMAX	new_dst = umax(old_dst, src0)
0Dh	AOP_UMIN	new_dst = umin(old_dst, src0)
Others	Reserved	Ignored

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Integer Trinary Operation Message Descriptor Control Field

MDC_AOP3 - Atomic Integer Trinary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x0000000E

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Format: Enumeration Specifies the atomic integer trinary operation to be performed

Value	Name	Description
00h	AOP_CMPWR_2W	new_dst = (src0_2W == old_dst_2W) ? src1_2W : old_dst_2W
0Eh	AOP_CMPWR **[Default]**	new_dst = (src0 == old_dst) ? src1 : old_dst
Others	Reserved	Ignored

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Integer Unary Operation Message Descriptor Control Field

MDC_AOP1 - Atomic Integer Unary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x00000005

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Format: Enumeration Specifies the atomic integer unary operation to be performed

Value	Name	Description
05h	AOP_INC **[Default]**	new_dst = old_dst + 1
06h	AOP_DEC	new_dst = old_dst - 1
0Fh	AOP_PREDEC	new_dst = old_dst - 1
Others	Reserved	Ignored

Programming Notes
When Return Data Control is set, new_dst is returned by AOP_PREDEC and otherwise old_dst is returned.

Audio Power State Format

Audio Power State Format		
Source:	BSpec	
Size (in bits):	2	
Default Value:	0x00000003	

DWord	Bit	Description
0	1:0	**Power State** Value / Name / Description 00b / D0 / D0 01b,10b / Unsupported / Unsupported 11b / D3 **[Default]** / D3

AVC CABAC

AVC CABAC		
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Reserved** Format: MBZ
	14	**Coefficient level out-of-bound Error** This flag indicates the coded coefficient level SEs in the bit-stream is out-of-bound.
	13	**Reserved** Format: MBZ
	12	**Reserved** Format: MBZ
	11	**Temporal Direction Motion Vector Out-of-Bound Error** This flag indicates motion vectors calculated from Temporal Direct Motion Vector is larger than the allowed range specified by the AVC spec.
	10	**Reserved** MBZ
	9	**Motion Vector Delta SE Out-of-Bound Error** This flag indicates inconsistent Motion Vector Delta SEs coded in the bit-stream.
	8	**Reference Index SE Out-of-Bound Error** This flag indicates inconsistent Reference Index SEs coded in the bit-stream.
	7	**MacroBlock QpDelta Error** This flag indicates out-of-bound MB QP delta SEs coded in the bit-stream.
	6	**Motion Vector Delta SE Error** This flag indicates out-of-bound motion vector delta SEs coded in the bit-stream.
	5	**Reference Index SE Error** This flag indicates out-of-bound Refidx SEs coded in the bit-stream.
	4	**Residual Error** This flag indicates out-of-bound absolute coefficient level SEs coded in the bit-stream.
	3	**Slice end Error** This flag indicates a pre-matured slice_end SE or inconsistent slice end on the last MB of a slice.
	2	**Chroma Intra prediction Mode Error** This flag indicates inconsistent Chroma Intra prediction mode SEs coded in the bit-stream.
	1	**Luma Intra prediction Mode Error** This flag indicates inconsistent luma Intra prediction mode SE coded in the bit-stream.
	0	**MB Concealment Flag** Each pulse from this flag indicates one MB is concealed by hardware.

AVC CAVLC

AVC CAVLC		
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Total Zero out-of-bound Error** This flag indicates the Total zero SE count exceed the max number of coeffs allowed in an intra16x16 AC block.
	14	**Coefficient level out-of-bound Error** This flag indicates the coded coefficient level SEs in the bit-stream is out-of-bound.
	13	**RunBefore out-of-bound Error** This flag indicates the coded RunBefore SE value is larger than the remaining zero block count.
	12	**Total coefficient Out-of-bound Error** This flag indicates the coded total coeff SE count exceed the max number of coeffs allowed in an intra16x16 AC block.
	11	**Temporal Direction Motion Vector Out-of-Bound Error** This flag indicates motion vectors calculated from Temporal Direct Motion Vector is larger than the allowed range specified by the AVC spec.
	10	**Reserved**
	9	**Motion Vector Delta SE Out-of-Bound Error** This flag indicates inconsistent Motion Vector Delta SEs coded in the bit-stream.
	8	**Reference Index SE Out-of-Bound Error** This flag indicates inconsistent Reference Index SEs coded in the bit-stream.
	7	**RunBefore/TotalZero Error** This flag indicates one or more inconsistent RunBefore or TotalZero SEs coded in the bit-stream.
	6	**Exponential Golomb Error** This flag indicates hardware detects more than 18 leadzero for skip and more than 19 for other SEs from the Exponential Golomb Logic
	5	**Total Coeff SE Error** This flag indicates one or more inconsistent total coeff SEs coded in the bit-stream.
	4	**Macroblock Coded Block Pattern Error** This flag indicates inconsistent CBP SEs coded in the bit-stream.
	3	**Mbytpe/submbtype Error** This flag indicates inconsistent MBtype/SubMBtype SEs coded in the bit-stream.
	2	**Chroma Intra prediction Mode Error** This flag indicates inconsistent Chroma Intra prediction mode SEs coded in the bit-stream.
	1	**Luma Intra prediction Mode Error** This flag indicates inconsistent luma Intra prediction mode SE coded in the bit-stream.

AVC CAVLC		
	0	**MB Concealment Flag** Each pulse from this flag indicates one MB is concealed by hardware.

AVS_Inline_DMEM

AVS_Inline_DMEM		
Source:	BSpec	
Size (in bits):	768	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:3	**Reserved** Format: MBZ
	2:0	**hw_used_bits** Index into the first valid bit of the starting byte of the first macroblock of the frame
1	31:14	**Reserved** Format: MBZ
	13:0	**hw_width** AVS SE: horizontal_size
2	31:14	**Reserved** Format: MBZ
	13:0	**hw_height** AVS SE: vertical_size
3	31:0	**hw_MbMax** In the VideoSequenceHeader of the AVS standard specification: MbWidth*MbHeight
4	31:2	**Reserved** Format: MBZ
	1:0	**hw_chroma_format** AVS SE: chroma_format
5	31:2	**Reserved** Format: MBZ
	1:0	**hw_picture_coding_type** AVS SE: picture_coding_type
6	31:8	**Reserved** Format: MBZ
	7:0	**hw_picture_distance** AVS SE: picture_distance
7	31:1	**Reserved** Format: MBZ

AVS_Inline_DMEM		
	0	**hw_picture_structure** AVS SE: picture_structure
8	31:0	**Reserved** Format: MBZ
9	31:0	**Reserved** Format: MBZ
10	31:0	**Reserved** Format: MBZ
11	31:0	**Reserved** Format: MBZ
12	31:0	**Reserved** Format: MBZ
13	31:0	**Reserved** Format: MBZ
14	31:0	**Reserved** Format: MBZ
15	31:0	**Reserved** Format: MBZ
16	31:0	**Reserved** Format: MBZ
17	31:0	**Reserved** Format: MBZ
18	31:1	**Reserved** Format: MBZ
	0	**hw_fixed_picture_qp** AVS SE: fixed_picture_qp
19	31:6	**Reserved** Format: MBZ
	5:0	**hw_picture_qp** AVS SE: picture_qp
20	31:1	**Reserved** Format: MBZ
	0	**hw_picture_reference_flag** AVS SE: picture_reference_flag
21	31:1	**Reserved** Format: MBZ

AVS_Inline_DMEM		
	0	**hw_skip_mode_flag** AVS SE: skip_mode_flag
22	31:1	**Reserved** Format: MBZ
	0	**hw_loop_filter_diable** AVS SE: loop_filter_disable
23	31:0	**Reserved** Format: MBZ

BaseAddress4KByteAligned

BaseAddress4KByteAligned		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Specifies a 64-bit (48-bit canonical) 4K-byte aligned memory base address.		
DWord	**Bit**	**Description**
0..1	63:12	**Base Address** Format: GraphicsAddress63-12
	11:0	**Reserved** Format: MBZ

BCS Hardware-Detected Error Bit Definitions

BCS Hardware-Detected Error Bit Definitions

Source: BlitterCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: **Value** / **Name** / **Description**: 1 / / Instruction Error detected **Programming Notes**: This error indications cannot be cleared except by reset (i.e., it is a fatal error).

BINDING_TABLE_EDIT_ENTRY

BINDING_TABLE_EDIT_ENTRY		
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Format: MBZ
	23:16	**Binding Table Index** Format: U8 This field specifies the index of binding table entry that will be updated.
	15:0	**Surface State Pointer** Format: SurfaceStateOffset[21:6]RENDER_SURFACE_STATE Surface State Pointer. This address points to a surface state block. This pointer is relative to the Surface State Base Address.

BINDING_TABLE_STATE

BINDING_TABLE_STATE		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
The binding table binds surfaces to logical resource indices used by shaders and other compute engine kernels. It is stored as an array of up to 256 elements, each of which contains one dword as defined here. The start of each element is spaced one dword apart. The first element of the binding table is aligned to a 64-byte boundary. Binding table indexes beyond 256 will automatically be mapped to entry 0 by the HW, w/ the exception of any messages which support the special indexes 240 through 255, inclusive.		
DWord	**Bit**	**Description**
0	31:6	**Surface State Pointer** Format: SurfaceStateOffset[31:6] This 64-byte aligned address points to a surface state block. This pointer is relative to the **Surface State Base Address**
	5:0	**Reserved** Format: MBZ

Bit Definition for Interrupt Control Registers - Blitter

Bit Definition for Interrupt Control Registers - Blitter		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:28	**Reserved** Format: MBZ These bits may be assigned to interrupts on future products/steppings.
	27	**Wait on Semaphore** Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait. Ring Buffer Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful.
	26:25	**Reserved** Format: MBZ
	24	**Context Switch Interrupt** Set when a context switch has just occurred. Exec-List Enable bit needs to be set for this interrupt to occur.
	23	**Reserved** Format: MBZ
	22	**Timeout Counter Expired** Set when the Blitter Command Streamer timeout counter has reached the timeout threshold value. This feature is not POR and SW must never unmask (enable) this interrupt via BCS_IMR.
	21	**Reserved** Format: MBZ
	20	**MI_FLUSH_DW Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	19	**Blitter Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error:** The Blitter Instruction Parser encounters an error while parsing an instruction.

Bit Definition for Interrupt Control Registers - Blitter		
	18:17	**Reserved** Format: MBZ
	16	**Blitter Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Blitter Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
	15:0	**Reserved** Format: MBZ

Bit Definition for Interrupt Control Registers - Media#1

Bit Definition for Interrupt Control Registers - Media#1

Source: VideoCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:12	**Reserved** Format: MBZ These bits may be assigned to interrupts on future products/steppings.
	11	**Wait on Semaphore** Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait. Ring Buffer Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful.
	10	**Reserved** Format: MBZ
	9	**Reserved**
	8	**Context Switch Interrupt** Set when a context switch has just occurred. **Execlist Enable bit** needs to be set for this interrupt to occur.
	7	**Reserved** Format: MBZ
	6	**Timeout Counter Expired** Set when the VCS timeout counter has reached the timeout thresh-hold value.
	5	**Reserved**
	4	**MI_FLUSH_DW Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	3	**Video Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error.

Bit Definition for Interrupt Control Registers - Media#1		
		Instruction Parser Error: The Blitter Instruction Parser encounters an error while parsing an instruction.
	2:1	**Reserved** Format: MBZ
	0	**Video Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

Bit Definition for Interrupt Control Registers - Media#2

Bit Definition for Interrupt Control Registers - Media#2		
Source:	VideoCS2	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:28	**Reserved** Format: MBZ These bits may be assigned to interrupts on future products/steppings.
	27	**Wait on Semaphore** Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait. Ring Buffer Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful.
	26	**Reserved** Format: MBZ
	25	**Reserved**
	24	**Context Switch Interrupt** Set when a context switch has just occurred. **Execlist Enable bit** needs to be set for this interrupt to occur.
	23	**Reserved** Format: MBZ
	22	**Timeout Counter Expired** Set when the VCS timeout counter has reached the timeout thresh-hold value.
	21	**Reserved**
	20	**MI_FLUSH_DW Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	19	**Video Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error:** The Blitter Instruction Parser encounters an error while parsing an instruction.

Bit Definition for Interrupt Control Registers - Media#2		
	18:17	**Reserved** Format: MBZ
	16	**Video Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.
	15:0	**Reserved** Format: MBZ

Bit Definition for Interrupt Control Registers - Render

Bit Definition for Interrupt Control Registers - Render

Source: RenderCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ Reserved for other command streamers - cannot be allocated by main command streamer.
	15:12	**Reserved** Format: MBZ
	11	**Wait on Semaphore** Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.
	10	**L3 Counter Save Interrupt**
	9	**TR Invalid Tile Detection** Hardware generates an interrupt when tiled resources translations hit an INVALID tile.
	8	**Context Switch Interrupt** Set when a context switch has just occurred. Execlist Enable bit needs to be set for this interrupt to occur.
	7	**Page Fault** Description: This interrupt is for handling Legacy Page Fault interface for all Command Streamers (BCS, RCS, VCS, VECS). When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c "Page Fault Support" section for more details.
	6	**Timeout Counter Expired** Set when the render pipe timeout counter (0x02190) has reached the timeout threshold value (0x0217c).
	5	**Reserved** Format: MBZ
	4	**PIPE_CONTROL Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	3	**Render Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit

Bit Definition for Interrupt Control Registers - Render		
		contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error:** The Render Instruction Parser encounters an error while parsing an instruction.
	2	**Reserved** Format: MBZ
	1	**Reserved**
	0	**Render Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

Bit Definition for Interrupt Control Registers - Video Enhancement

Bit Definition for Interrupt Control Registers - Video Enhancement

Source: VideoEnhancementCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:12	**Reserved** Format: MBZ These bits may be assigned to interrupts on future products/steppings.
	11	**Wait on Semaphore** Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait. Ring Buffer Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful.
	10	**Reserved** Format: MBZ
	9	**Reserved** Format: MBZ
	8	**Context Switch Interrupt** Set when a context switch has just occurred. Exec-List Enable bit needs to be set for this interrupt to occur.
	7	**Reserved** Format: MBZ
	6	**Timeout Counter Expired** Set when the Video Enhancement Command Streamer timeout counter has reached the timeout threshold value. This feature is not POR and SW must never unmask (enable) this interrupt via VECS_IMR.
	5	**Reserved** Format: MBZ
	4	**MI_FLUSH_DW Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	3	**Video Enhancement Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be

Bit Definition for Interrupt Control Registers - Video Enhancement		
		set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error:** The Blitter Instruction Parser encounters an error while parsing an instruction.
	2:1	**Reserved** Format: MBZ
	0	**Video Enhancement Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Video Enhancement Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

BLEND_STATE

BLEND_STATE

Source:	BSpec
Size (in bits):	544
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

The blend state is stored as a structure containing a common DWORD that applies to all RTs and an array of up to 8 elements, each of which contains the two DWords for each. The start of each element is spaced 2 DWords apart. The blend state is aligned to a 64-byte boundary, which is pointed to by a field in 3DSTATE_BLEND_STATE_POINTERS. The 3-bit Render Target Index field in the Render Target Write data port message header is used to select which of the 8 elements from BLEND_STATE that is used on the current message.

DWord	Bit	Description
0	31	**Alpha To Coverage Enable** Format: Enable If set, Source0 Alpha is converted to a temporary 1/2/4-bit coverage mask and the mask bit corresponding to the sample# ANDed with the sample mask bit. If set, sample coverage is computed based on src0 alpha value. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels.The field is applied to all the RTs in MRT case.
	30	**Independent Alpha Blend Enable** Format: Enable When enabled, the other fields in this instruction control the combination of the alpha components in the Color Buffer Blend stage. When disabled, the alpha components are combined in the same fashion as the color components.The field is applied to all the RTs in MRT case.
	29	**Alpha To One Enable** Format: Enable If set, Source0 Alpha is set to 1.0f after (possibly) being used to generate the AlphaToCoverage coverage mask.If Dual Source Blending is enabled, this bit must be disabled.The field is applied to all the RTs in MRT case.
	28	**Alpha To Coverage Dither Enable** Format: Enable If set, sample coverage is computed based on src0 alpha value and it modulates the sample coverage based on screen coordinates. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any

BLEND_STATE

		value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels. If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.The field is applied to all the RTs in MRT case.
	27	**Alpha Test Enable** Format: Enable Enables the AlphaTest function of the Pixel Processing pipeline.The field is applied to all the RTs in MRT case. **Programming Notes** Alpha Test can only be enabled if Pixel Shader outputs a float alpha value. Alpha Test is applied independently on each render target by comparing that render target's alpha value against the alpha reference value. If the alpha test fails, the corresponding pixel write will be supressed only for that render target. The depth/stencil update will occur if alpha test passes for any render target.
	26:24	**Alpha Test Function** Format: 3D_Compare_Function This field specifies the comparison function used in the AlphaTest functionThe field is applied to all the RTs in MRT case.
	23	**Color Dither Enable** Format: Enable Enables dithering of colors (including any alpha component) before they are written to the Color Buffer. The field is applied to all the RTs in MRT case. **Programming Notes** For YUV render target formats, this field must be programmed to 0.
	22:21	**X Dither Offset** Format: U2 Specifies offset to apply to pixel X coordinate LSBs when accessing dither table.The field is applied to all the RTs in MRT case.
	20:19	**Y Dither Offset** Format: U2 Specifies offset to apply to pixel Y coordinate LSBs when accessing dither table.The field is applied to all the RTs in MRT case.
	18:0	**Reserved** Format: MBZ
1..16	63:0	**Entry** Format: BLEND_STATE_ENTRY

BLEND_STATE_ENTRY

BLEND_STATE_ENTRY

Source: BSpec
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63	**Logic Op Enable** Format: Enable Enables the LogicOp function of the Pixel Processing pipeline. **Programming Notes** Enabling LogicOp and Color Buffer Blending at the same time is UNDEFINED
	62:59	**Logic Op Function** Format: 3D_Logic_Op_Function This field specifies the function to be performed (when enabled) in the Logic Op stage of the Pixel Processing pipeline. Note that the encoding of this field is one less than the corresponding "R2_" ROP code defined in WINGDI.H, and is a rather contorted mapping of the OpenGL LogicOp encodings. However, this field was defined such that, when the 4 bits are replicated to 8 bits, they coincide with the ROP codes used in the Blter. Note: if the Logic Op Function does not depend on "D", the dest buffer is not read.
	58:37	**Reserved** Format: MBZ
	36	**Pre-Blend Source Only Clamp Enable** Format: Enable This field specifies whether the source(s) are clamped prior to blending, regardless of whether blending is enabled. If DISABLED, no clamping is performed prior to blending. If ENABLED, only source0 and source 1, if dual source is enabled, are clamped prior to the blend to the range specified by Color Clamp Range. **Value / Name / Description** 0 / Disabled / No clamping is performed prior to blending. 1 / Enabled / Only Source(s) are clamped prior to blend function. Other inputs to blend must be clamped according to the behavior specified for "pre-blend color clamp disable" in the pre-blend color clamping table . **Programming Notes** This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. **When this bit is enabled Pre-Blend Color Clamp Enable must be disabled.**

BLEND_STATE_ENTRY

35:34 **Color Clamp Range**

Specifies the clamped range used in Pre-Blend and Post-Blend Color Clamp functions if one or both of those functions are enabled. Note that this range selection is shared between those functions. This field is ignored if both of the Color Clamp Enables are disabled

Value	Name	Description
0	COLORCLAMP_UNORM	Clamp Range [0,1]
1	COLORCLAMP_SNORM	Clamp Range [-1,1]
2	COLORCLAMP_RTFORMAT	Clamp to the range of the RT surface format (Note: The Alpha component is clamped to FLOAT16 for R11G11B10_FLOAT format). Unsigned Floating Point components are clamped to positive zero.
3	Reserved	Reserved

33 **Pre-Blend Color Clamp Enable**

Format:	Enable

This field specifies whether the source, destination and constant color channels are clamped prior to blending, regardless of whether blending is enabled.If DISABLED, no clamping is performed prior to blending.If ENABLED, all inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.

Value	Name	Description
0	Disabled	No clamping is performed prior to blending.
1	Enabled	All inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.

Programming Notes
See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.

32 **Post-Blend Color Clamp Enable**

Format:	Enable

Regardless of whether this clamping is enabled, the blending output channels will be clamped to the RT surface format just prior to being written.

Programming Notes
This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. Post Blend Clamp Enable must be programmed identical to Pre Blend Clamp Enable. The device will automatically clamp source color channels to the respective RT surface range. **When this bit is enabled Pre-Blend Source Only Clamp Enable must be disabled.**

BLEND_STATE_ENTRY

	31	**Color Buffer Blend Enable** Format: Enable Enables the ColorBufferBlending (nee "alpha blending") function of the Pixel Processing Pipeline for this render target. **Programming Notes** Enabling LogicOp and ColorBufferBlending at the same time is UNDEFINED
	30:26	**Source Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Controls the "source factor" in the ColorBufferBlending function.Refer to Source Alpha Blend Factor for encodings.
	25:21	**Destination Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Controls the "destination factor" in the ColorBufferBlending function. Refer to Source Alpha Blend Factor for encodings.
	20:18	**Color Blend Function** Format: 3D_Color_Buffer_Blend_Function This field specifies the function used to combine the color components in the ColorBufferBlending function of the Pixel Processing Pipeline. If Independent Alpha Blend Enable is disabled, this field will also control the blending of the alpha components in the ColorBufferBlending function.
	17:13	**Source Alpha Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Controls the "source factor" in alpha Color Buffer Blending stage.Note: For the source/destination alpha blend factors, the encodings indicating "COLOR" are the same as the encodings indicating "ALPHA", as the alpha component of the color is selected.
	12:8	**Destination Alpha Blend Factor** Format: 3D_Color_Buffer_Blend_Factor Controls the "destination factor" in alpha Color Buffer Blending stage. Refer to Source Alpha Blend Factor for encodings.
	7:5	**Alpha Blend Function** Format: 3D_Color_Buffer_Blend_Function This field specifies the function used to combine the alpha components in the Color Buffer blend stage of the Pixel Pipeline when the IndependentAlphaBlend state is enabled.

BLEND_STATE_ENTRY

	4	**Reserved** Format: MBZ
	3	**Write Disable Alpha** Format: Disable This field controls the writing of the alpha component into the Render Target. **Value** / **Name** / **Description** 0b / Enabled / Alpha component can be overwritten 1b / Disabled / Writes to the color buffer will not modify Alpha. **Programming Notes** For YUV surfaces, this field must be set to 0B (enabled).
	2	**Write Disable Red** Format: Disable This field controls the writing of the red component into the Render Target. **Value** / **Name** / **Description** 0b / Enabled / Red component can be overwritten 1b / Disabled / Writes to the color buffer will not modify Red. **Programming Notes** For YUV surfaces, this field must be set to 0B (enabled).
	1	**Write Disable Green** Format: Disable This field controls the writing of the green component into the Render Target. **Value** / **Name** / **Description** 0b / Enabled / Green component can be overwritten 1b / Disabled / Writes to the color buffer will not modify Green. **Programming Notes** For YUV surfaces, this field must be set to 0B (enabled).

BLEND_STATE_ENTRY

0 **Write Disable Blue**

Format:	Disable

This field controls the writing of the Blue component into the Render Target.

Value	Name	Description
0b	Enabled	Blue component can be overwritten
1b	Disabled	Writes to the color buffer will not modify Blue.

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

Block Dimensions Message Header Control

MHC_BDIM - Block Dimensions Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:22	**Reserved** Format: Ignore Ignored
	21:20	**Block Height** Format: Enumeration Height in rows of block being accessed. Range = [0,3] representing 1 to 8 rows.

Value	Name	Description
0h	H1	Block height = 1 row
1h	H2	Block height = 2 rows
2h	H4	Block height = 4 rows
03h	H8	Block height = 8 rows

DWord	Bit	Description
	19:2	**Reserved** Format: Ignore Ignored
	1:0	**Block Width** Format: Enumeration Width in Dwords of block being accessed. Range = [0,3] representing 1 to 8 Dwords.

Value	Name	Description
0h	W1	Block width = 1 Dword
1h	W2	Block width = 2 Dwords
2h	W4	Block width = 4 Dwords
03h	W8	Block width = 8 Dwords

Block Message Header

MH_BTS_GO - Block Message Header

Source: DataPort 0

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**Reserved** Format: Ignore Ignored
2	31:0	**Global Offset** Format: U32 Specifies the global element index into the buffer, in units of Hwords, Owords, Dwords, or Bytes (depending on the message). **Programming Notes** The Global Offset for the Aligned Block operations is specified as a Dword-aligned byte offset (offset bits [1:0] = 0), Oword-aligned byte offset (offset bits [3:0]=0), or Hword-aligned byte offset (offset bits [4:0]=0). If the address offset calculated with the Global Offset is greater than the Surface Size, then the access is Out-of-Bounds.
3..7	159:0	**Reserved** Format: Ignore Ignored

BR00 - BLT Opcode and Control

BR00 - BLT Opcode and Control

Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**BLT Engine Busy** This bit indicates whether the BLT Engine is busy (1) or idle (0). This bit is replicated in the SETUP BLT Opcode and Control register. Value: 0 — Name: Idle **[Default]** Value: 1 — Name: Busy
	30	**Setup Instruction Instruction** Default Value: 0 The current instruction performs clipping (1).
	29	**Setup Monochrome Pattern** This bit is decoded from the Setup instruction opcode to identify whether a color (0) or monochrome (1) pattern is used with the SCANLINE_BLT instruction. Value: 0 — Name: Color **[Default]** Value: 1 — Name: Monochrome
	28:22	**Instruction Target (Opcode)** Default Value: 0000000b This is the contents of the Instruction Target field from the last BLT instruction. This field is used by the BLT Engine state machine to identify the BLT instruction it is to perform. The opcode specifies whether the source and pattern operands are color or monochrome.
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value: 00b — Name: **[Default]** Value: 1xb — Name: Write Alpha Channel Value: x1b — Name: Write RGB Channel

BR00 - BLT Opcode and Control

19:17 Monochrome Source Start

Default Value:	000b

This field indicates the starting monochrome pixel bit position within a byte per scan line of the source operand. The monochrome source is word aligned which means that at the end of the scan line all bits should be discarded until the next word boundary.

16 Bit/Byte Packed

Byte packed is for the NT driver.

Value	Name
0b	Bit **[Default]**
1b	Byte

15 Src Tiling Enable

Value	Name
0b	Tiling Disabled (Linear) **[Default]**
1b	Tiling enabled: Tile-X or Tile-Y

14:12 Horizontal Pattern Seed

Default Value:	0b

This field indicates the pattern pixel position which corresponds to X = 0.

11 Dest Tiling Enable

When set to '1', this means that Blitter is executing in Tiled mode. If '0' it means that Blitter is in Linear mode. Pre-Dev Blitter never executes in Tiled-Y mode, DevGT+ Blitter supports both Tile-X and Tile-Y modes. On reset, this bit will be '0'. This definition applies to only X, Y Blits.

Value	Name
0b	Tiling Disabled (Linear blit) **[Default]**
1b	Tiling enabled: Tile-X or Tile-Y

10:8 Transparency Range Mode

These bits control whether or not the byte(s) at the destination corresponding to a given pixel will be conditionally written, and what those conditions are. This feature can make it possible to perform various masking functions in order to selectively write or preserve graphics data already at the destination.

Value	Name	Description
xx0b	**[Default]**	No color transparency mode enabled. This causes normal operation with regard to writing data to the destination.

BR00 - BLT Opcode and Control

		001b		[Source color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (R, G, B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
		011b		[Source and Alpha color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (A, R, G, B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation."
		101b		[Destination and Alpha color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (A, R, G, B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
		111b		[Destination color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (R, G, B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
	7:5	**Pattern Vertical Seed**		
		Default Value:	000b	
		This field specifies the pattern scan line which corresponds to Y=0.		
	4	**Destination Read Modify Write**		
		Default Value:	0b	
		This bit is decoded from the last instruction's opcode field and Destination Transparency Mode to identify whether a Destination read is needed.		

BR00 - BLT Opcode and Control

	3	**Color Source** Default Value: 0b This bit is decoded from the last instructions opcode field to identify whether a color (1) source is used.
	2	**Monochrome Source** Default Value: 0b This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) source is used.
	1	**Color Pattern** Default Value: 0b This bit is decoded from the last instructions opcode field to identify whether a color (1) pattern is used.
	0	**Monochrome Pattern** Default Value: 0b This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) pattern is used.

BR01 - Setup BLT Raster OP, Control, and Destination Offset

BR01 - Setup BLT Raster OP, Control, and Destination Offset

Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Solid Pattern Select** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations.
1b		The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.

	30	**Clipping Enabled**

Value	Name
0b	**[Default]**
1b	

	29	**Monochrome Source Transparency Mode** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1b		Wherever a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

BR01 - Setup BLT Raster OP, Control, and Destination Offset

28 **Monochrome Pattern Transparency Mode**

This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the pattern data. Wherever a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1b		Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

27:26 **32bpp Byte Mask**

This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field.

Value	Name
00b	**[Default]**
1xb	Write Alpha Channel
x1b	Write RGB Channel

25:24 **Color Depth**

Value	Name
00b	8 Bit Color Depth **[Default]**
01b	16 Bit Color Depth
10b	Alternate 16 Bit Color Depth
11b	32 Bit Color Depth

23:16 **Raster Operation Select**

These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine.

15:0 **Destination Pitch (Offset)**

For non-XY Blits, the signed 16bit field allows for specifying upto + 32Kbytes signed pitches in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Destination will be 512Byte aligned and should be programmable upto + 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Destination will be 128Byte aligned and should be programmable upto + 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto + 32KDWords. For X, Y

BR01 - Setup BLT Raster OP, Control, and Destination Offset		
		blits with nontiled surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto + 32Kbytes (same as before). These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

BR05 - Setup Expansion Background Color

BR05 - Setup Expansion Background Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Setup Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. BR05 is also used as the solid pattern for the PIXEL_BLT instruction. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR06 - Setup Expansion Foreground Color

BR06 - Setup Expansion Foreground Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Setup Expansion Foreground Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR07 - Setup Blit Color Pattern Address Lower Order Address bits

BR07 - Setup Blit Color Pattern Address Lower Order Address bits

Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:6	**Setup Blit Color Pattern Address** Format: GraphicsAddress[31:6] Lower 32bits of the 48bit addressing. These 26 bits specify the starting address of the (8X8) pixel color **pattern from the SETUP_BLT instruction.** This register works identically to the Pattern Address register (BR15), but this version is **only used with the SCANLINE_BLT instruction execution** (the actual programming for this, is done in XY_SETUP_BLT command). The pattern data must be located in linear memory. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and is supplied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
	5:0	**Reserved** Format: MBZ

BR09 - Destination Address Lower Order Address Bits

BR09 - Destination Address Lower Order Address Bits		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Destination Address Bits** Format: GraphicsAddress[31:0] When tiling is enabled for XY-blits, this base address should be limited to 4KB. when tiling is disabled for XY-blits, this base address should be CL (64byte) aligned. These lower 32bits of the 48bit address, which specify the starting pixel address of the destination data. This register is also the working destination address register for the lower 32bits of the address, and changes as the BLT Engine performs the accesses. Used as the scan line address (Destination Y Address and Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 and Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text. Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), this address points to the first byte to be written. Note: Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

BR11 - BLT Source Pitch (Offset)

BR11 - BLT Source Pitch (Offset)		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:16	**Reserved**
	15:0	**Source Pitch (Offset)** For non-XY Blits with color source operand (SRC_COPY_BLT), the signed 16bit field allows for specifying upto + 32Kbytes signed pitch in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Color Source will be 512Byte aligned and should be programmable upto + 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Color Source will be 128Byte aligned and should be programmable upto + 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto + 32KDWords. For X, Y blits with nontiled color source surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto + 32Kbytes (same as before). When the color source data is located within the frame buffer or AGP aperture, these signed 16 bits store the memory address offset (pitch) value by which the source address originally specified in the Source Address Register is incremented or decremented as each scan line's worth of source data is read from the frame buffer by the BLT Engine, so that the source address will point to the next memory address from which the next scan line's worth of source data is to be read. Note that if the intended source of a BLT operation is within on-screen frame buffer memory, this offset is normally set to accommodate the fact that each subsequent scan line's worth of source data lines up vertically with the source data in the scan line, above. However, if the intended source of a BLT operation is within off-screen memory, this offset can be set to accommodate a situation in which the source data exists as a single contiguous block of bytes where in each subsequent scan line's worth of source data is stored at a location immediately after the location where the source data for the last scan line ended.

BR12 - Source Address Lower order Address bits

BR12 - Source Address Lower order Address bits		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Source Address Bits** Format: GraphicsAddress[31:0] Lower 32bits of the 48bit addressing. When tiling is enabled for XY-blits with Color source surfaces, this base address should be aligned to 4KB. When tiling is disabled for XY-blits, this base address should be CL (64byte) aligned. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read. These lower 32bits of the 48bit address, specify the starting pixel address of the color source data. The lower 3 bits are used to indicate the position of the first valid byte within the first Quadword of the source data. If this Source happens to be a Monosource surface, then this Monosource Base Address programmed, must always be Cache Line (64byte) aligned.

BR13 - BLT Raster OP, Control, and Destination Pitch

BR13 - BLT Raster OP, Control, and Destination Pitch

Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Solid Pattern Select** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations.
1		The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.

30	**Clipping Enabled**
Default Value:	0

29	**Monochrome Source Transparency Mode** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1		Where a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

BR13 - BLT Raster OP, Control, and Destination Pitch

28 **Monochrome Pattern Transparency Mode**

This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode in the Opcode and Control register.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the pattern data. Where a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1		Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

27:26 **32bpp Byte Mask**

This field is only used for 32bpp.

Value	Name
00b	**[Default]**
1xb	Write Alpha Channel
x1b	Write RGB Channel

25:24 **Color Depth**

Value	Name
00b	8 Bit Color Depth **[Default]**
01b	16 Bit Color Depth
10b	24 Bit Color Depth
11b	Reserved

23:16 **Raster Operation Select**

Default Value:	00000000b

These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine.

BR13 - BLT Raster OP, Control, and Destination Pitch		
	15:0	**Destination Pitch(Offset)** These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

BR14 - Destination Width and Height

BR14 - Destination Width and Height		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
BR14 contains the values for the height and width of the data to be BLT. If these values are not correct, such that the BLT Engine is either expecting data it does not receive or receives data it did not expect, the system can hang.		
DWord	**Bit**	**Description**
0	31:29	**Reserved**
	28:16	**Destination Height** These 13 bits specify the height of the destination data in terms of the number of scan lines. This is a working register.
	15:13	**Reserved**
	12:0	**Destination Byte Width** These 13 bits specify the width of the destination data in terms of the number of bytes per scan line. The number of pixels per scan line into which this value translates depends upon the color depth to which the graphics system has been set.

BR15 - Color Pattern Address Lower order Address bits

BR15 - Color Pattern Address Lower order Address bits

Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:6	**Color Pattern Address** Format: GraphicsAddress[31:6] Lower 32bits of the 48bit addressing. There is no change to the Color Pattern address specification due to Non-Power-of-2 change. It remains the same as before. The pattern data must be located in linear memory. These 26 bits specify the starting address of the (8X8) pixel color pattern. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and are applied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
	5:0	**Reserved** Format: MBZ

BR16 - Pattern Expansion Background and Solid Pattern Color

BR16 - Pattern Expansion Background and Solid Pattern Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR17 - Pattern Expansion Foreground Color

BR17 - Pattern Expansion Foreground Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR18 - Source Expansion Background and Destination Color

BR18 - Source Expansion Background and Destination Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Source Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome source data during BLT operations. This register is also used to support destination transparency mode and Solid color fill. Whether one, two, three, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR19 - Source Expansion Foreground Color

BR19 - Source Expansion Foreground Color		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern/Source Expansion Foreground Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome source data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR27 - Destination Higher Order Address

BR27 - Destination Higher Order Address

Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

Upper 32 bits of the starting pixel address for the destination data. This structure is also the working location for the upper bits of the destination address, and changes as the BLT Engine performs the accesses. See BR09 for the lower 32 bits. When tiling is enabled for XY-blits, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before.

Used as the scan line address (Destination Y Address and Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 and Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text.

Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), the destination address points to the first byte to be written. This structure is always the last location written for a BLT drawing instruction. Writing to BR27 starts the BLT engine execution. Note: Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost reserved bits are ignored and MBZ.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Destination Address Upper DWORD** Format: GraphicsAddress[47:32]

BR28 - Source Higher Order Address

BR28 - Source Higher Order Address			
Source:	BlitterCS		
Size (in bits):	32		
Default Value:	0x00000000		
DWord		**Bit**	**Description**
0 Upper 32 bits of the Source address, specifying the starting pixel address of the color or mono source data. When tiling is enabled for XY-blits with Color source surfaces, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before, including for monosource and text blits. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost reserved bits are ignored and MBZ.		31:16	**Reserved** Format: MBZ
		15:0	**Source Address Upper DWORD** Format: GraphicsAddress[47:32]

BR29 - Color Pattern Higher Order Address

BR29 - Color Pattern Higher Order Address			
Source:	BlitterCS		
Size (in bits):	32		
Default Value:	0x00000000		
DWord		**Bit**	**Description**
0 Upper 32 bits of the Color Pattern address, specifying the starting location of the (8X8) pixel pattern. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost reserved bits are ignored and MBZ.		31:16	**Reserved** Format: MBZ
		15:0	**Color Pattern Address Upper DWORD** Format: GraphicsAddress[47:32]

BR30 - Setup Blit Color Pattern Higher Order Address

BR30 - Setup Blit Color Pattern Higher Order Address		
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0 Upper 32 bits of the Color Pattern address, specifying the starting location of the (8X8) pixel pattern. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost reserved bits are ignored and MBZ.	31:16	**Reserved** Format: MBZ
	15:0	**Setup Blit Color Pattern Upper DWORD** Format: GraphicsAddress[47:32]

Byte Masked Media Block Message Header

MH_MBBM - Byte Masked Media Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** Must be DWord aligned (Bits 1:0 MBZ) for the write form of the message.
1	31:0	**Y Offset** Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Media Block Message Control** Format: MHC_MBBM_CONTROL Specifies the Byte Masked message subtype and its additional input parameters.
3	31:0	**Byte Mask** Format: U32 Specifies the Byte Mask for writes when Message Mode field is BYTE_MASK. **Programming Notes** The Byte mask applies horizontally to each row of output: bit 0 for byte 0, through bit 31 for byte 31.
4	31:0	**FFTID** Format: MHC_FFTID Fixed Function Thread ID
5..7	95:0	**Reserved** Format: Ignore Ignored

Byte Masked Media Block Message Header Control

MHC_MBBM_CONTROL - Byte Masked Media Block Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Format: Enumeration Specifies the Media Block Write Message subtype is Byte Masked. Value / Name / Description: 02h / BYTE_MASK / The Block Height and Block Width fields are specified in this Dword. The Byte Mask qualifies which bytes are written. Others / Reserved / Reserved.
	29	**Reserved** Format: Ignore Ignored
	28:24	**Sub-Register Offset** Format: U5 This field is ignored (reserved) for Media Block Write message.
	23:22	**Reserved** Format: Ignore Ignored
	21:16	**Block Height** Format: U6 Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows Restriction Restriction : If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) * (# rows) <= 64 Dwords.
	15:10	**Reserved** Format: Ignore Ignored

MHC_MBBM_CONTROL - Byte Masked Media Block Message Header Control

	9:8	**Register Pitch Control** Format: U2 This field is ignored (reserved) for a Media Block Write message.
	7:6	**Reserved** Format: Ignore Ignored
	5:0	**Block Width** Format: U6 Width in bytes of the block being accessed. Range = [0,31] representing 1 to 32 Bytes. **Programming Notes** Must be DWord aligned for Media Block Write message.

CC_VIEWPORT

CC_VIEWPORT

Source: BSpec
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

The viewport state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary. The Minimum and Maximum Depth legal value ranges are dependant on the depth buffer format.

DWord	Bit	Description
0	31:0	**Minimum Depth** Format: IEEE_Float Indicates the minimum depth. The interpolated or computed depth is clamped to this value prior to the depth test. **Programming Notes** The Minimum depth value must be less-than-or-equal to the Maximum depth value. The Minimum depth value cannot be NAN (Not-A-Number). For All depth formats:Minimum depth value must not be less than 0.0, also it may not be -0.0 (negative zero)
1	31:0	**Maximum Depth** Format: IEEE_Float Indicates the maximum depth. The interpolated or computed depth is clamped to this value prior to the depth test. **Programming Notes** The Maximum depth value cannot be smaller than Minimum depth value. The Maximum depth value cannot be NAN (Not-A-Number). For all depth formats: The Maximum depth value must be between +0.0to +1.0.

Channel Mask Message Descriptor Control Field

MDC_CMASK - Channel Mask Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3:0	**Mask** Format: Enumeration For the read message, indicates that which channels are read from the surface and included in the writeback message. For the write message, indicates which channels are included in the message payload and written to the surface.

Value	Name	Description
00h	RGBA **[Default]**	Red, Green, Blue, and Alpha are included
01h	GBA	Green, Blue, and Alpha are included
02h	RBA	Red, Blue, and Alpha are included
03h	BA	Blue and Alpha are included
04h	RGA	Red, Green, and Alpha are included
05h	GA	Green and Alpha are included
06h	RA	Red and Alpha are included
07h	A	Alpha is included
08h	RGB	Red, Green, and Blue are included
09h	GB	Green and Blue are included
0Ah	RB	Red and Blue are included
0Bh	B	Blue is included
0Ch	RG	Red and Green are included
0Dh	G	Green is included
0Eh	R	Red is included
0Fh	Reserved	Ignored

Channel Mode Message Descriptor Control Field

MDC_CMODE - Channel Mode Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Channel Mode** Format: Enumeration Two modes of channel-enable are provided: a SIMD8 or SIMD16 Dword channel serial view of a register, and a SIMD4x2 view of a register.

Value	Name	Description
0	Oword	All 4 Dwords are read or written if one or more of these channels are enabled
1	Dword	Each Dword is read or written only if its corresponding channel is enabled.

Clock Gating Disable Format

Clock Gating Disable Format		
Source:	BSpec	
Size (in bits):	1	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	0	**Clock_Gate_Disable**

Value	Name	Description
0b	Enable	Clock gating controlled by unit logic
1b	Disable	Disable clock gating function

Clock Gating Disable Format

Clock Gating Disable Format

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Clock Gate Disable**

Value	Name	Description
0b	Enable	Clock gating controlled by unit enabling logic
1b	Disable	Disable clock gating function

COLOR_CALC_STATE

COLOR_CALC_STATE

Source: BSpec
Size (in bits): 192
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This definition is pointed to by a field in 3DSTATE_CC_STATE_POINTERS, and stored at a 64-byte aligned boundary.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15	**Round Disable Function Disable** Format: Disable Disables the round-disable function of the color calculator. Value 0 – Name: Cancelled – Description: Dithering is cancelled based on the data used by blend to avoid drift. Value 1 – Name: Not Cancelled – Description: Dithering is NOT cancelled.
	14:1	**Reserved** Format: MBZ
	0	**Alpha Test Format** This field selects the format for Alpha Reference Value and the format in which Alpha Test is performed. Value 0h – Name: ALPHATEST_UNORM8 – Description: UNorm8 Value 1h – Name: ALPHATEST_FLOAT32 – Description: Float32 **Programming Notes** Alpha-test format is independent of RT format. When PS outputs UNIT/SINT alpha-value, it will be treated as IEEE 32bit float number for the purpose of alpha-test.
1	31:0	**Alpha Reference Value As UNORM8** Exists If: [Alpha Test Format] == 'ALPHATEST_UNORM8' Format: UNORM8 Upper 24 bits MBZ This field specifies the alpha reference value to compare against in the Alpha Test function.
	31:0	**Alpha Reference Value As FLOAT32** Exists If: [Alpha Test Format] == 'ALPHATEST_FLOAT32' Format: IEEE_Float This field specifies the alpha reference value to compare against in the Alpha Test function.

COLOR_CALC_STATE		
2	31:0	**Blend Constant Color Red** Format: IEEE_Float This field specifies the Red channel of the Constant Color used in Color Buffer Blending.
3	31:0	**Blend Constant Color Green** Format: IEEE_Float This field specifies the Green channel of the Constant Color used in Color Buffer Blending.
4	31:0	**Blend Constant Color Blue** Format: IEEE_Float This field specifies the Blue channel of the Constant Color used in Color Buffer Blending.
5	31:0	**Blend Constant Color Alpha** Format: IEEE_Float This field specifies the Alpha channel of the Constant Color used in Color Buffer Blending.

COLOR_PROCESSING_STATE - ACE State

COLOR_PROCESSING_STATE - ACE State

Source: BSpec

Size (in bits): 416

Default Value: 0x00000068, 0x4C382410, 0x9C887460, 0xEBD8C4B0, 0x604C3824, 0xB09C8874, 0x0000D8C4, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the ACE state used by the color processing function.
It corresponds to DW29..DW41 of the Color Processing State.

DWord	Bit	Description
0	31:7	**Reserved** Format: MBZ
	6:2	**Skin Threshold** Format: U5 Used for Y analysis (min/max) for pixels which are higher than skin threshold. Value / Name: 1-31 26 **[Default]**
	1	**Full Image Histogram** Default Value: 0 Format: Enable Used to ignore the area of interest for full image histogram.
	0	**ACE Enable** Format: Enable
1	31:24	**Y3** Default Value: 76 Format: U8 The value of the y_pixel for point 3 in PWL.
	23:16	**Y2** Default Value: 56 Format: U8 The value of the y_pixel for point 2 in PWL.

COLOR_PROCESSING_STATE - ACE State

	15:8	**Y1** Default Value: 36 Format: U8 The value of the y_pixel for point 1 in PWL.
	7:0	**Ymin** Default Value: 16 Format: U8 The value of the y_pixel for point 0 in PWL.
2	31:24	**Y7** Default Value: 156 Format: U8 The value of the y_pixel for point 7 in PWL.
	23:16	**Y6** Default Value: 136 Format: U8 The value of the y_pixel for point 6 in PWL.
	15:8	**Y5** Default Value: 116 Format: U8 The value of the y_pixel for point 5 in PWL.
	7:0	**Y4** Default Value: 96 Format: U8 The value of the y_pixel for point 4 in PWL.
3	31:24	**Ymax** Default Value: 235 Format: U8 The value of the y_pixel for point 11 in PWL.
	23:16	**Y10** Default Value: 216 Format: U8 The value of the y_pixel for point 10 in PWL.

COLOR_PROCESSING_STATE - ACE State

	15:8	**Y9** Default Value: 196 Format: U8 The value of the y_pixel for point 9 in PWL.
	7:0	**Y8** Default Value: 176 Format: U8 The value of the y_pixel for point 8 in PWL.
4	31:24	**B4** Default Value: 96 Format: U8 The value of the bias for point 4 in PWL.
	23:16	**B3** Default Value: 76 Format: U8 The value of the bias for point 3 in PWL.
	15:8	**B2** Default Value: 56 Format: U8 The value of the bias for point 2 in PWL.
	7:0	**B1** Default Value: 36 Format: U8 The value of the bias for point 1 in PWL.
5	31:24	**B8** Default Value: 176 Format: U8 The value of the bias for point 8 in PWL.
	23:16	**B7** Default Value: 156 Format: U8 The value of the bias for point 7 in PWL.

COLOR_PROCESSING_STATE - ACE State		
	15:8	**B6** Default Value: 136 Format: U8 The value of the bias for point 6 in PWL.
	7:0	**B5** Default Value: 116 Format: U8 The value of the bias for point 5 in PWL.
6	31:16	**Reserved** Format: MBZ
	15:8	**B10** Default Value: 216 Format: U8 The value of the bias for point 10 in PWL.
	7:0	**B9** Default Value: 196 Format: U8 The value of the bias for point 9 in PWL.
7	31:27	**Reserved** Format: MBZ
	26:16	**S1** Format: U1.10 The value of the slope for point 1 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S0** Format: U1.10 The value of the slope for point 0 in PWL. The default is 1024/1024.
8	31:27	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - ACE State		
	26:16	**S3** Format: U1.10 The value of the slope for point 3 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S2** Format: U1.10 The value of the slope for point 2 in PWL. The default is 1024/1024.
9	31:27	**Reserved** Format: MBZ
	26:16	**S5** Format: U1.10 The value of the slope for point 5 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S4** Format: U1.10 The value of the slope for point 4 in PWL. The default is 1024/1024.
10	31:27	**Reserved** Format: MBZ
	26:16	**S7** Format: U1.10 The value of the slope for point 7 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S6** Format: U1.10 The value of the slope for point 6 in PWL. The default is 1024/1024.

COLOR_PROCESSING_STATE - ACE State		
11	31:27	**Reserved** Format: MBZ
	26:16	**S9** Format: U1.10 The value of the slope for point 9 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S8** Format: U1.10 The value of the slope for point 8 in PWL. The default is 1024/1024.
12	31:11	**Reserved** Format: MBZ
	10:0	**S10** Format: U1.10 The value of the slope for point 10 in PWL. The default is 1024/1024.

COLOR_PROCESSING_STATE - CSC State

COLOR_PROCESSING_STATE - CSC State

Source: BSpec

Size (in bits): 288

Default Value: 0x00002000, 0x00000000, 0x00000400, 0x00000000, 0x000004B4, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the CSC state used by the color processing function.
It corresponds to DW55..DW63 of the Color Processing State.

DWord	Bit	Description
0	31:29	**Reserved** Format: MBZ
	28:16	**C1** Default Value: 0 Format: S2.10 2's complement Transform coefficient
	15:3	**C0** Default Value: 1024 Format: S2.10 2's complement Transform coefficient
	2	**YUV_IN** Default Value: 0 Format: YUV CSC input offset enable.
	1	**YUV_OUT** Default Value: 0 Format: RGB CSC output offset enable.
	0	**Transform Enable** Format: Enable
1	31:26	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - CSC State

	25:13	**C3** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C2** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
2	31:26	**Reserved** Format: MBZ
	25:13	**C5** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C4** Default Value: 1024 Format: S2.10 2's complement Transform coefficient.
3	31:26	**Reserved** Format: MBZ
	25:13	**C7** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C6** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
4	31:13	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - CSC State

	12:0	**C8** Default Value: 1204 Format: S2.10 2's complement Transform coefficient.
5	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 1** Default Value: 0 Format: S9 2's complement Offset Out for Y/R.
	9:0	**Offset In 1** Default Value: 0 Format: S9 2's complement Offset in for Y/R.
6	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 2** Default Value: 0 Format: S9 2's complement Offset out for U/G.
	9:0	**Offset in 2** Default Value: 0 Format: S9 2's complement Offset in for U/G.
7	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 3** Default Value: 0 Format: S9 2's complement Offset out for V/B.

COLOR_PROCESSING_STATE - CSC State

	9:0	**Offset in 3** Default Value: 0 Format: S9 2's complement Offset in for V/B.
8	31:17	**Reserved** Format: MBZ
	16	**Alpha from State Select** Format: U1 Enumerated Type Value / Name / Description 0 / Alpha is taken from message 1 / Alpha is taken from state
	15:0	**Color Pipe Alpha** Format: U16

COLOR_PROCESSING_STATE - PROCAMP State

COLOR_PROCESSING_STATE - PROCAMP State

Source: BSpec
Size (in bits): 64
Default Value: 0x00020001, 0x01000000

This state structure contains the PROCAMP state used by the color processing function.
It corresponds to DW53..DW54 of the Color Processing State.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:17	**Contrast** Default Value: 1 Format: U4.7 Contrast magnitude.
	16:13	**Reserved** Format: MBZ
	12:1	**Brightness** Default Value: 0 Format: S7.4 2's complement Brightness magnitude.
	0	**PROCAMP Enable** Default Value: 1 Format: Enable
1	31:16	**Cos_c_s** Default Value: 256 Format: S7.8 2's complement UV multiplication cosine factor.
	15:0	**Sin_c_s** Default Value: 0 Format: S7.8 2's complement UV multiplication sine factor.

COLOR_PROCESSING_STATE - STD/STE State

COLOR_PROCESSING_STATE - STD/STE State

Source: BSpec

Size (in bits): 928

Default Value: 0x9A6E39F0, 0x400C0000, 0x00001180, 0xFE2F2E00, 0x000000FF, 0x00140000, 0xD82E0000, 0x8285ECEC, 0x00008282, 0x00000000, 0x02117000, 0xA38FEC96, 0x00008CC8, 0x00000000, 0x01478000, 0x0007C300, 0x00000000, 0x00000000, 0x1C180000, 0x00000000, 0x00000000, 0x00000000, 0x0007CF80, 0x00000000, 0x00000000, 0x1C080000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the STD/STE state used by the color processing function.

DWord	Bit	Description
0	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate
	15:10	**Hue Max** Default Value: 14 Format: U6 Rectangle half width
	9:4	**Sat Max** Default Value: 31 Format: U6 Rectangle half length.
	3	**Reserved** Format: MBZ
	2	**Output Control** 0 – Output Pixels **[Default]** 1 – Output STD Decisions

COLOR_PROCESSING_STATE - STD/STE State

	1	**STE Enable** Format: Enable
	0	**STD Enable** Format: Enable
1	31	**Reserved** Format: MBZ
	30:28	**Diamond Margin** Default Value: 4 Format: U3
	27:21	**Diamond du** Default Value: 0 Format: S6 2's complement Rhombus center shift in the sat-direction, relative to the rectangle center.
	20:18	**HS Margin** Default Value: 3 Format: U3
	17:10	**Cos(α)** Format: S0.7 2's Compliment The default is 79/128
	9:8	**Reserved** Format: MBZ
	7:0	**Sin(α)** Format: S0.7 2's Compliment The default is 101/128
2	31:21	**Reserved** Format: MBZ
	20:13	**Diamond Alpha** Format: U2.6 1 / tan(β) The default is 100/64

COLOR_PROCESSING_STATE - STD/STE State

	12:7	**Diamond Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.
	6:0	**Diamond dv** Default Value: 0 Format: S6 2's complement
3	31:24	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	23:16	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	15:8	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
	7	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	6:0	**Reserved** Format: MBZ
4	31:18	**Reserved** Format: MBZ
	17:13	**Y_Slope_2** Format: U2.3 Slope between points Y3 and Y4. The default is 31/8.

COLOR_PROCESSING_STATE - STD/STE State

	12:8	**Y_Slope_1** Format: U2.3 Slope between points Y1 and Y2. The default is 31/8.
	7:0	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function
5	31:16	**INV_skin_types_margin** Format: U0.16 1/(2* Skin_types_margin) Value: 20 – Name: **[Default]** – Description: Skin_Type_margin
	15:0	**Inverse Margin VYL** Format: U0.16 1 / Margin_VYL The default is 3300/65536
6	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**Inverse Margin VYU** Format: U0.16 1 / Margin_VYU The default is 1600/65536.
7	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.

COLOR_PROCESSING_STATE - STD/STE State

	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y point 2 of the lower part of the detection PWLF.
8	31:27	**Reserved** Format: MBZ
	26:16	**S0L** Format: S2.8 2's complement Slope 0 of the lower part of the detection PWLF. The default is -5/256.
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8 V Bias 2 of the lower part of the detection PWLF.
9	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Format: S2.8 2's complement Slope 2 of the lower part of the detection PWLF. The default is 0/256.

COLOR_PROCESSING_STATE - STD/STE State		
	10:0	**S1L** Format: S2.8 2's complement Slope 1 of the lower part of the detection PWLF. The default is 0/256.
10	31:27	**Reserved** Format: MBZ
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Format: S2.8 2's complement Slope 3 of the lower part of the detection PWLF. The default is 0/256.
11	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.

COLOR_PROCESSING_STATE - STD/STE State		
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
12	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Format: S2.8 2's complement Slope 0 of the upper part of the detection PWLF. The default is 256/256.
	15:8	**B3U** Default Value: 140 Format: U8 V Bias 3 of the upper part of the detection PWLF.
	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
13	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Format: S2.8 2's complement Slope 2 of the upper part of the detection PWLF. The default is -179/256.
	10:0	**S1U** Format: S2.8 2's complement Slope 1 of the upper part of the detection PWLF. The default is -113/256.
14	31:28	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - STD/STE State

	27:20	**Skin Types Margin** Default Value: 20 Format: U8 Skin types Y margin.
	19:12	**Skin Types Thresh** Default Value: 120 Format: U8 Skin types Y threshold.
	11	**Skin Type Enable** Format: Enable Treat differently bright and dark skin types. 0 **[Default]** Disable
	10:0	**S3U** Format: S2.8 2's complement Slope 3 of the upper part of the detection PWLF. The default is 0/256.
15	31	**Reserved** Format: MBZ
	30:21	**SATB1** Format: S7.2 2's complement First bias for the saturation PWLF (bright skin). The default is -8/4.
	20:14	**SATP3** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (bright skin).
	13:7	**SATP2** Default Value: 6 Format: S6 2's complement Second point for the saturation PWLF (bright skin).

COLOR_PROCESSING_STATE - STD/STE State

	6:0	**SATP1** Format: S6 2's complement First point for the saturation PWLF (bright skin). The default is -6.
16	31	**Reserved** Format: MBZ
	30:20	**SATS0** Format: U3.8 Zeroth slope for the saturation PWLF (bright skin). The default is 297/256.
	19:10	**SATB3** Format: S7.2 2's complement Third bias for the saturation PWLF (bright skin). The default is 124/4.
	9:0	**SATB2** Format: S7.2 2's complement Second bias for the saturation PWLF (bright skin). The default is 8/4.
17	31:22	**Reserved** Format: MBZ
	21:11	**SATS2** Format: U3.8 Second slope for the saturation PWLF (bright skin). The default is 297/256.
	10:0	**SATS1** Format: U3.8 First slope for the saturation PWLF (bright skin). The default is 85/256.
18	31:25	**HUEP3** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (bright skin)

COLOR_PROCESSING_STATE - STD/STE State

	24:18	**HUEP2** Default Value: 6 Format: S6 2's complement Second point for the hue PWLF (bright skin)
	17:11	**HUEP1** Format: S6 2's complement First point for the hue PWLF (bright skin). The default is -6.
	10:0	**SATS3** Format: U3.8 Thrid slope for the saturation PWLF (bright skin). The default is 256/256.
19	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3** Format: S7.2 2's complement Third bias for the hue PWLF (bright skin). The default is 56/4.
	19:10	**HUEB2** Format: S7.2 2's complement Second bias for the hue PWLF (bright skin). The default is 8/4.
	9:0	**HUEB1** Format: S7.2 2's complement First bias for the hue PWLF (bright skin). The default is -8/4.
20	31:22	**Reserved** Format: MBZ
	21:11	**HUES1** Format: U3.8 First slope for the hue PWLF (bright skin) The default is 85/256.

COLOR_PROCESSING_STATE - STD/STE State

	10:0	**HUES0** Format: U3.8 Zeroth slope for the hue PWLF (bright skin) The default is 384/256.
21	31:22	**Reserved** Format: MBZ
	21:11	**HUES3** Format: U3.8 Third slope for the hue PWLF (bright skin) The default is 256/256.
	10:0	**HUES2** Format: U3.8 Second slope for the hue PWLF (bright skin) The default is 384/256.
22	31	**Reserved** Format: MBZ
	30:21	**SATB1_DARK** Format: S7.2 2's complement First bias for the saturation PWLF (dark skin) The default is 0/4.
	20:14	**SATP3_DARK** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (dark skin)
	13:7	**SATP2_DARK** Default Value: 31 Format: S6 2's complement Second point for the saturation PWLF (dark skin)
	6:0	**SATP1_DARK** Format: S6 2's complement First point for the saturation PWLF (dark skin). The default is -11.

COLOR_PROCESSING_STATE - STD/STE State		
23	31	**Reserved** Format: MBZ
	30:20	**SATS0_DARK** Format: U3.8 Zeroth slope for the saturation PWLF (dark skin). The default is 397/256.
	19:10	**SATB3_DARK** Format: S7.2 2's complement Third bias for the saturation PWLF (dark skin). The default is 124/4.
	9:0	**SATB2_DARK** Format: S7.2 2's complement Second bias for the saturation PWLF (dark skin). The default is 124/4.
24	31:22	**Reserved** Format: MBZ
	21:11	**SATS2_DARK** Format: U3.8 Second slope for the saturation PWLF (dark skin). The default is 256/256.
	10:0	**SATS1_DARK** Format: U3.8 First slope for the saturation PWLF (dark skin). The default is 189/256.
25	31:25	**HUEP3_DARK** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	24:18	**HUEP2_DARK** Default Value: 2 Format: S6 2's complement Third point for the hue PWLF (dark skin).

COLOR_PROCESSING_STATE - STD/STE State

	17:11	**HUEP1_DARK** Default Value: 0 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	10:0	**SATS3_DARK** Format: U3.8 Third slope for the saturation PWLF (dark skin). The default is 256/256.
26	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3_DARK** Format: S7.2 2's complement Third bias for the hue PWLF (dark skin). The default is 56/4.
	19:10	**HUEB2_DARK** Format: S7.2 2's complement Second bias for the hue PWLF (dark skin). The default is 0/4.
	9:0	**HUEB1_DARK** Format: S7.2 2's complement First bias for the hue PWLF (dark skin). The default is 0/4.
27	31:22	**Reserved** Format: MBZ
	21:11	**HUES1_DARK** Format: U3.8 First slope for the hue PWLF (dark skin). The default is 0/256.
	10:0	**HUES0_DARK** Format: U3.8 Zeroth slope for the hue PWLF (dark skin). The default is 256/256.

COLOR_PROCESSING_STATE - STD/STE State		
28	31:22	**Reserved** Format: MBZ
	21:11	**HUES3_DARK** Format: U3.8 Third slope for the hue PWLF (dark skin). The default is 256/256.
	10:0	**HUES2_DARK** Format: U3.8 Second slope for the hue PWLF (dark skin). The default is 299/256.

COLOR_PROCESSING_STATE - TCC State

COLOR_PROCESSING_STATE - TCC State

Source: BSpec

Size (in bits): 352

Default Value: 0xDCDCDC00, 0xDCDCDC00, 0x1E34CC91, 0x3E3CCE91, 0x02E80195, 0x0197046B, 0x01790174, 0x00096000, 0x00000000, 0x03030000, 0x009201C0

This state structure contains the TCC state used by the color processing function.
It corresponds to DW42..DW52 of the Color Processing State.

DWord	Bit	Description
0	31:24	**SatFactor3** Default Value: 220 Format: U1.7 The saturation factor for yellow.
	23:16	**SatFactor2** Default Value: 220 Format: U1.7 The saturation factor for red.
	15:8	**SatFactor1** Default Value: 220 Format: U1.7 The saturation factor for magenta.
	7	**TCC Enable** Format: Enable
	6:0	**Reserved** Format: MBZ
1	31:24	**SatFactor6** Default Value: 220 Format: U1.7 The saturation factor for blue.
	23:16	**SatFactor5** Default Value: 220 Format: U1.7 The saturation factor for cyan.

COLOR_PROCESSING_STATE - TCC State		
	15:8	**SatFactor4** Default Value: 220 Format: U1.7 The saturation factor for green.
	7:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ
	29:20	**Base Color 3** Default Value: 483 Format: U10
	19:10	**Base Color 2** Default Value: 307 Format: U10
	9:0	**Base Color 1** Default Value: 145 Format: U10
3	31:30	**Reserved** Format: MBZ
	29:20	**Base Color 6** Default Value: 995 Format: U10
	19:10	**Base Color 5** Default Value: 819 Format: U10
	9:0	**Base Color 4** Default Value: 657 Format: U10
4	31:16	**Color Transit Slope 23** Default Value: 744 Format: U0.16 The calculation result of 1 / (BC3 - BC2) [1/62]

COLOR_PROCESSING_STATE - TCC State

	15:0	**Color Transit Slope 12** Default Value: 405 Format: U0.16 The calculation result of 1 / (BC2 - BC1) [1/57]
5	31:16	**Color Transit Slope 45** Default Value: 407 Format: U0.16 The calculation result of 1 / (BC5 - BC4) [1/57]
	15:0	**Color Transit Slope 34** Default Value: 1131 Format: U0.16 The calculation result of 1 / (BC4 - BC3) [1/61]
6	31:16	**Color Transit Slope 61** Default Value: 377 Format: U0.16 The calculation result of 1 / (BC1 - BC6) [1/62]
	15:0	**Color Transit Slope 56** Default Value: 372 Format: U0.16 The calculation result of 1 / (BC6 - BC5) [1/62]
7	31:22	**Color Bias 3** Default Value: 0 Format: U2.8 Color bias for BaseColor3.
	21:12	**Color Bias 2** Default Value: 150 Format: U2.8 Color bias for BaseColor2.
	11:2	**Color Bias 1** Default Value: 0 Format: U2.8 Color bias for BaseColor1.

COLOR_PROCESSING_STATE - TCC State

	1:0	**Reserved** Format: MBZ
8	31:22	**Color Bias 6** Default Value: 0 Format: U2.8 Color bias for BaseColor6.
	21:12	**Color Bias 5** Default Value: 0 Format: U2.8 Color bias for BaseColor5.
	11:2	**ColorBias4** Default Value: 0 Format: U2.8 Color bias for BaseColor4.
	1:0	**Reserved** Format: MBZ
9	31	**Reserved** Format: MBZ
	30:24	**UV Threshold** Default Value: 3 Format: U7 Low UV threshold.
	23:19	**Reserved** Format: MBZ
	18:16	**UV Threshold Bits** Default Value: 3 Format: U3 Low UV transition width bits.
	15:13	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - TCC State		
	12:8	**STE Threshold** Default Value: 0 Format: U5 Skin tone pixels enhancement threshold.
	7:3	**Reserved** Format: MBZ
	2:0	**STE Slope Bits** Default Value: 0 Format: U3 Skin tone pixels enhancement slope bits.
10	31:16	**Inverse UVMax Color** Default Value: 146 Format: U0.16 1 / UVMaxColor. Used for the SFs2 calculation.
	15:9	**Reserved** Format: MBZ
	8:0	**UVMax Color** Default Value: 448 Format: U9 The maximum absolute value of the legal UV pixels. Used for the SFs2 calculation.

Color Calculator State Pointer Message Header Control

MHC_RT_CCSP - Color Calculator State Pointer Message Header Control

Source:	BSpec
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:6	**Color Calculator State Pointer** Format: GeneralStateOffset[31:6] Specifies the 64-byte aligned point to the color calculator state. This pointer is relative to the General State Base Address.
	5:0	**Reserved** Format: Ignore Ignored

Color Code Message Header Control

MHC_RT_CC - Color Code Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:10	**Reserved** Format: Ignore Ignored
	9:8	**Color Code** Format: U2 This ID is assigned by the Windower unit and is used to track synchronizing events. Reserved for HW implementation use
	7:0	**FFTID** Format: U8 This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.

Context Descriptor Format

Context Descriptor Format

Source:	BSpec
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

This is the format of context descriptors which make up submitted execlists.

DWord	Bit	Description

0..1	63:32	**Context ID**
		Description
		Context ID is a unique field assigned by GFX driver when a new context is created by which it is identified across all hierarchies of SW and HW.
		• Context ID is used for semaphore signaling by hardware and software.
		• Context ID matching is used by hardware to detect Lite Restore.
		• Context ID is used by hardware for page fault reporting and response with IOMMU.
		• Context switch reason and the associated Context ID are reported to Context Switch Status Buffer by hardware on a context switch.
		Context ID[31:0] (bits[63:32] of the context descriptor) are used for comparing during lite restore, semaphore signaling and context specific OA enabling.
		Context ID which is a 32 bit field is further divided in to three segments described below:
		• **Bits[63:55] (Bits 31:23 of Context ID)** is referred to as GroupID. GroupId+PASID combination of a context must be a unique identifier for contexts that are active in the system. The definition of active context is listed as:
		  • Any Context that is already submitted to h/w or already running in h/w.
		  • Any Context that hit page faults, was preempted (didn't run to context complete), and is waiting to be resubmitted pending IOMMU "last in group" response.
		  • Any Context that has experienced reset but not all faults are responded to.
		• **Bit[54] (Bit 22 of Context ID)** – MBZ for SW programming; this bit is used by hardware to distinguish between F&H vs F&S page requests and response messages to and from IOMMU. This bit is used by hardware on receiving page response to properly manage the page fault counters
		• **Bit[53] (Bit 21 of Context ID)** – MBZ from SW programming, is reserved for future hardware use.
		• **Bits[52:32] (Bits 20:0 of Context ID)** are for software use-only and must be unique field assigned by GFX driver when a new context is created.
		Context ID is reported by hardware to OABUFFER along with the performance statistics counters, Context ID is used for filtering the statistics on per context basis.

Context Descriptor Format

	31:12	**Logical Ring Context Address (LRCA)** Format: GraphicsAddress[31:12] This field contains the 4 KB-aligned address of the Logical Ring Context associated with this execlist element. LRCA must be always programmed in GGTT memory.
	11:9	**Reserved** Format: MBZ
	8	**Privilege Access** This field when set indicates PPGTT enabled in legacy context mode. In advanced context mode this field is reserved and must be zero.
	7:6	**Fault Handling** Source: CommandStreamer

Value	Name	Description
0h	Fault and Hang	Fault model is not supported and fault occurrence is treated as catastrophic. GAM indicates Fault Error to Command streamer. Fault Error interrupt is reported to scheduler. Command Streamer will not initiate context switch on occurrence of Fault Error.
1h	Fault and Halt	In this mode of operation faults are detected and corrected in the Page Walker. The client unit is stalled until the fault is corrected. The command streamer cannot preempt the context until the page fault condition is corrected and the access is completed. Restriction : When both coherent memory accesses are present in L3 memory fabric and OS-managed SVM is enabled, there are TLB invalidate deadlock conditions that require Fault-and-Stream fault mode instead of Fault-and-Halt.
2h	Fault and Stream	In this mode of operation faults are allowed on EU memory accesses. Page Walker will directly work with memory page handler to fix the faults on the fly for these surfaces. Command streamer is not aware of the fault service being done by page walker and goes with its normal execution rules for context switch. On completion of flush during context switch CS explicitly requests acknowledge message to Page Walker before proceeding further. Page Walker acknowledges Command Streamer once it is done on a clean boundary. Page Walker asserts Fault Error on occurrence of non recoverable fault or access violations (Command Streamer access, VFunit access, etc) to Command Streamer; this is the same as Fault and Hang behavior.
Others	Reserved	Reserved

Programming Notes
When execlist mode is set to "Legacy Context mode" Fault Handling mode must be set to "Fault and Hang."

Context Descriptor Format

		For proper programming for Page Fault modes, refer to memory interface section of the Bspec for the corresponding generation.
	5	**Reserved** Format: MBZ
	4:3	**Addressing Mode & Legacy Context** Format: U2 Legacy context set indicates GPU is operating in legacy context mode of operation and doesn't support any SVM features. Legacy context reset indicates GPU is operating in advanced context mode of operation and support SVM features. Based on the Context mode set Addressing mode is interpreted appropriately. The table below summarizes the combinations supported. GFX engine always uses 32b virtual addressing mode when translated using GGTT irrespective of below options.
	2	**Force Restore** Setting this bit will force a context restore operation when switching to this context even if the LRCA in the CCID register (normally the LRCA of the last context from the prior execlist) matches this one. Note that it is legal (and likely desirable) for the **Render Context Restore Inhibit** bit (part of the CTXT_SR_CTL register) in the context image being restored to also be set. The "ring" context is being forced to be restored from a newly initialized context despite a possible LRCA match. However, the render context for such a newly initialized context will likely be uninitialized and so should not be restored.

Value	Name	Description
00b	Advanced Context with no A/D support	GPU is enabled for advanced context mode and supports SVM features. GPU DOESN'T support Access and Dirty bit management in page tables. GPU supports 64b(48bit canonical) PPGTT graphics virtual addressing. PDP0_DESCRIPTOR contains the PASID (process address space identifier) and other PDP Descriptors are ignored.
01b	Legacy Context with no 64 bit VA support	GPU is enabled for legacy context mode of operation and DOESN'T support any SVM features. GPU supports 32b PPGTT graphics virtual addressing. PDP*_DESCRIPTOR contains the base address to 4GB of memory space supported.
10b	Advanced Context with A/D support	GPU is enabled for advanced context mode and supports SVM features. GPU DOES support Access and Dirty bit management in page tables. GPU supports 64b (48bit canonical) PPGTT graphics virtual addressing. PDP0_DESCRIPTOR contains the PASID (process address space identifier) and other PDP Descriptors are ignored.
11b	Legacy Context with 64 bit VA support	GPU is enabled for legacy context mode of operation and DOESN'T support any SVM features. GPU supports 64b (48bit canonical) PPGTT graphics virtual addressing and PDP0_DESCRIPTOR contains the base address to PML4 and other PDP Descriptors are ignored.

Context Descriptor Format		
	1	**Force PD Restore** Setting this bit will cause the on-chip page directory to be reloaded from the PD image in memory even on an LRCA match. No other operations of context restore will occur on an LRCA match, however. Software should set this bit if it has updated a context's page directory and wants the context to begin using the new page directory without having to switch away from it (to another context) and back again. Setting this bit will have no effect if **Force Restore** is also set; a complete context restore (including the PD) will be performed.
	0	**Valid** Set if this register holds a valid context descriptor. SW should set this bit in the Element registers that it has set up to contain valid context descriptors. Any execlist elements that are not used in a submitted execlist must have this bit clear.

Context Status

Context Status

Source: BSpec
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0	63:32	**Context ID** Format: U32 The format of Context ID (sub-fields) described in "Context Descriptor Format".
	31:30	**Reserved** Format: MBZ
	29	**Reserved** Format: MBZ
	28	**Reserved**
	27:25	**Reserved** Format: MBZ
	24:20	**Reserved** Format: MBZ
	19:16	**Display Plane** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set.

Value	Name
0h	Reserved (Look at field 14:12)
1h	Reserved
2h	Reserved
3h	Display Plane-7
4h	Display Plane-8
5h	Display Plane-9
6h	Display Plane-10
7h	Display Plane-11
8h	Display Plane-12
[9h, Fh]	Reserved

Context Status

15 — Lite Restore

Format:	Enable

This bit is only valid only when Preempted bit is set.
When set, this bit indicates that a given context got preempted with the same context resulting in Lite Restore in HW.

14:12 — Display Plane Additional

This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch.
This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set.
(Future - could remove the Sprites and move to bits 19:16)

Value	Name	Exists If
0h	Display Plane-1	[Wait on V-blank]==0
0h	Display Pipe-A	[Wait on V-blank]==1
1h	Display Plane-2	[Wait on V-blank]==0
1h	Display Pipe-B	[Wait on V-blank]==1
2h	Display Plane-3	[Wait on V-blank]==0
2h	Display Pipe-C	[Wait on V-blank]==1
3h	Display Plane-4	
4h	Display Plane-5	
5h	Display Plane-6	

11 — Semaphore Wait Mode

Value	Name
0h	Signal Mode
1h	Poll Mode

10:9 — Reserved

Format:	MBZ

8 — Wait on Scanline

7 — Wait on Semaphore

6 — Wait on V-blank

5 — Wait on Sync Flip

4 — Context Complete
Element is completely processed (Head eqv to Tail) and resulted in a context switch.

3 — ACTIVE to IDLE
Following this context switch there is no active element available in HW to execute

Context Status		
	2	**Element Switch** Context Switch happened from first element in the current execlist to the second element of the same execlist
	1	**Preempted** Submission of a new execlist has resulted in context switch. The switch is from element in current execlist to element in pending execlist
	0	**IDLE to ACTIVE** Execlist submitted when HW is IDLE. When this bit is set rest of the fields in CSQ are not valid.

CSC COEFFICIENT FORMAT

CSC COEFFICIENT FORMAT

Source: BSpec
Size (in bits): 16
Default Value: 0x00000000

Coefficients for the CSC are stored in sign-exponent-mantissa format.
Two CSC coefficients are stored in each dword, the table below show the data packing in each dword.

DWord	Bit	Description
0	15	**Sign** Value: 0b — Name: Positive Value: 1b — Name: Negative
	14:12	**Exponent_bits** Represented as 2^(-n) Value: 110b — Name: 4 — Description: 4 or mantissa is bb.bbbbbbb Value: 111b — Name: 2 — Description: 2 or mantissa is b.bbbbbbbb Value: 000b — Name: 1 — Description: 1 or mantissa is 0.bbbbbbbbb Value: 001b — Name: 0.5 — Description: 0.5 or mantissa is 0.0bbbbbbbbb Value: 010b — Name: 0.25 — Description: 0.25 or mantissa is 0.00bbbbbbbbb Value: 011b — Name: 0.125 — Description: 0.125 or mantissa is 0.000bbbbbbbbb Value: Others — Name: Reserved — Description: Reserved
	11:3	**Mantissa**
	2:0	**Reserved**

Data Port 0 Message Types

MT_DP0 - Data Port 0 Message Types

Source: DataPort 0
Size (in bits): 5
Default Value: 0x00000000

Lists all the Message Types in a Data Port 0 Message Descriptor [18:14].
The Legacy messages are encoded in Data Port 0 with Bit 18 set to zero. The Message Header is optional for many (but not all) of these operations.
The Scratch Block messages are encoded in Data Port 0 with Bit 18 set to one. A Message Header is required.

DWord	Bit	Description
0	4	**Legacy DAP-DC Message** Format: Enumeration Legacy Message

Value	Name	Description
0h	No **[Default]**	Legacy DAP-DC Message
1h	Reserved	Scratch Block Message, descriptor uses different Message Type encoding

DWord	Bit	Description
0	3:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description
00h	MT0R_B **[Default]**	Block Read message
01h	MT0R_AB	Aligned Block Read message
02h	MT0R_OWDB	Oword Dual Block Read message
03h	MT0R_DWS	Dword Scattered Read message
04h	MT0R_BS	Byte Scattered Read message
07h	MT0_MEMFENCE	Memory Fence message
08h	MT0W_B	Block Write message
0Ah	MT0W_OWDB	Oword Dual Block Write message
0Bh	MT0W_DWS	Dword Scattered Write message
0Ch	MT0W_BS	Byte Scattered Write message
Others	Reserved	Ignored

Data Port 1 Message Types

MT_DP1 - Data Port 1 Message Types

Source: DataPort 1

Size (in bits): 5

Default Value: 0x00000000

Lists all the Message Types in a Data Port 1 Message Descriptor [18:14].
Most surface and atomic operations, both typed and untyped, are encoded on Data Port 1. The Message Header is optional for many (but not all) of these operations.
Most A64 Stateless operations are also encoded on Data Port 1. The Message Header is forbidden for all A64 messages on Data Port 1.

DWord	Bit	Description
0	4:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description
00h	MT1R_T **[Default]**	Transpose Read message
01h	MT1R_US	Untyped Surface Read message
02h	MT1A_UI	Untyped Atomic Integer Operation message
03h	MT1A_UI4x2	Untyped Atomic Integer Operation SIMD4x2 message
04h	MT1R_MB	Media Block Read message
05h	MT1R_TS	Typed Surface Read message
06h	MT1A_TA	Typed Atomic Integer Operation message
07h	MT1A_TA4x2	Typed Atomic Integer Operation SIMD4x2 message
08h	Reserved	Ignored
09h	MT1W_US	Untyped Surface Write mesage
0Ah	MT1W_MB	Media Block Write message
0Bh	MT1A_TC	Typed Atomic Counter Operation message
0Ch	MT1A_TC4x2	Typed Atomic Counter Operation SIMD4x2 message
0Dh	MT1W_TS	Typed Surface Write message
0Eh	Reserved	Ignored
10h	MT1R_A64_SB	A64 Scattered Read message
11h	MT1R_A64_US	A64 Untyped Surface Read message
12h	MT1A_A64_UI	A64 Untyped Atomic Integer Operation message
13h	MT1A_A64_UI4x2	A64 Untyped Atomic Integer Operation SIMD4x2 message
14h	MT1R_A64_B	A64 Block Read message
15h	MT1W_A64_B	A64 Block Write message

MT_DP1 - Data Port 1 Message Types

		18h	Reserved	Ignored
		19h	MT1W_A64_US	A64 Untyped Surface Write message
		1Ah	MT1W_A64_SB	A64 Scattered Write message
		1Bh	MT1A_UF	Untyped Atomic Float Operation message
		1Ch	MT1A_UF4x2	Untyped Atomic Float Operation SIMD4x2 message
		1Dh	MT1A_A64_UF	A64 Untyped Atomic Float Operation message
		1Eh	MT1A_A64_UF4x2	A64 Untyped Atomic Float Operation SIMD4x2 message
		Others	Reserved	Ignored

Data Port 2 Extended Message Descriptor

DP2_EXTDESC - Data Port 2 Extended Message Descriptor

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Sideband Offset** Format: U16 Specifies the 16-bit offset from the Sideband added to all the offsets in the Address Payload for DP2 messages.
	15:11	**Reserved** Format: MBZ Ignored
	10:0	**Execution Unit Extended Message Descriptor Definition** Format: Execution_Unit_Extended_Message_Descriptor EU uses this information as part of the SEND instruction.

Data Port 2 Message Types

MT_DP2 - Data Port 2 Message Types

Source: DataPort 2

Size (in bits): 5

Default Value: 0x00000002

Lists all the Message Types in a Data Port 2 Message Descriptor [18:14]. Scaled operations are on Data Port 2. They provide a pitch-scaled data address calculation for SLM Stateless address models. The Message Header is forbidden for SLM operations.

DWord	Bit	Description
0	4:1	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description
01h	MT2R_US **[Default]**	Untyped Surface Read message
02h	MT2R_A64_SB	A64 Scattered Read message
03h	MT2R_A64_US	A64 Untyped Surface Read message
04h	MT2R_BS	Byte Scattered Read message
09h	MT2W_US	Untyped Surface Write message
0Ah	MT2W_A64_US	A64 Untyped Surface Write message
0Bh	MT2W_A64_SB	A64 Scattered Write message
0Ch	MT2W_BS	Byte Scattered Write message
Others	Reserved	Ignored

	0	**Reserved** Format: MBZ Ignored

Data Port Bindless Surface Extended Message Descriptor

DP_EXTDESC_BTI252 - Data Port Bindless Surface Extended Message Descriptor

Source:	BSpec
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:12	**Bindless Surface Offset** Format: BindlessSurfaceOffset[25:6] Specifies the bindless surface offset if the Binding Table Index is set to 252. Ignored otherwise. The bindless surface offset is added to the Bindless Surface Base Address as bits 25:6 of the byte-based address. The resulting address is the location of SURFACE_STATE for this message.
	11	**Reserved**
	10:0	**Execution Unit Extended Message Descriptor Definition** Format: Execution_Unit_Extended_Message_Descriptor EU uses this information as part of the SEND instruction.

Data Size Message Descriptor Control Field

MDC_DS - Data Size Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**Data Size** Format: Enumeration Specifies the number of Bytes to be read or written

Value	Name	Description
00h	B	1 Byte
01h	W	2 Bytes
02h	DW	4 Bytes
03h	Reserved	Reserved

Deptrh Clear Value Format

STRUCTURE_TEMPLATE - Deptrh Clear Value Format

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:0	**Address1** Format: IEEE_FP Format: UNORM24 Format: UNORM16 When this field contains 24-bit UNORM, the upper 8-bits are reserved (0's) When this field contains 16-bit UNORM the upper 16-bits are reserved (0's)

Display Engine Render Response Message Definition

Display Engine Render Response Message Definition

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

The Display Engine Render Response Registers use bit definitions from this table.

Programming Notes

Some events can be sent to CS (Render Command Streamer) or BCS (Blitter Command Streamer). For render response messages sending flip done or scanline events, the destination, CS or BCS, is selected depending on the initiator of the flip or the load scanline command. For render response messages sending vertical blank events, the destinations, CS or BCS, or both CS and BCS,is selected depending on the DE_RR_DEST setting. Command Streamer Plane number to the Display Plane name mapping is available in the **Display Plane Capability and Interoperability section**.

The STEREO3D_EVENT_MASK selects between left eye and right eye reporting of vertical blank and scanline events in stereo 3D modes.

DWord	Bit	Description
0	31	**Reserved**
	30	**Reserved**
	29	**Reserved**
	28	**Reserved**
	27:23	**Reserved**
	22	**Reserved**
	21	**Pipe_C_Start_of_Vertical_Blank_Event** This event is reported on the start of the vertical blank of the transcoder attached to Pipe C.
	20	**Plane_2_C_Flip_Done_Event** This event is reported on the completion of a flip for Plane 2 C.
	19	**Reserved**
	18	**Plane_4_B_Flip_Done_Event** This event is reported on the completion of a flip for Plane 4 B. Not all projects have a plane 4 B.
	17	**Plane_4_A_Flip_Done_Event** This event is reported on the completion of a flip for Plane 4 A. Not all projects have a plane 4 A.
	16	**Plane_3_C_Flip_Done_Event** This event is reported on the completion of a flip for Plane 3 C.
	15	**Plane_1_C_Flip_Done_Event** This event is reported on the completion of a flip for Plane 1 C.

Display Engine Render Response Message Definition		
	14	**Pipe_C_Scanline_Event** This event is reported on the start of the selected scan line for the transcoder attached to Pipe C.
	13	**Reserved**
	12	**Reserved**
	11	**Pipe_B_Start_of_Vertical_Blank_Event** This event is reported on the start of the vertical blank of the transcoder attached to Pipe B.
	10	**Plane_2_B_Flip_Done_Event** This event is reported on the completion of a flip for Plane 2 B.
	9	**Plane_1_B_Flip_Done_Event** This event is reported on the completion of a flip for Plane 1 B.
	8	**Pipe_B_Scanline_Event** This event is reported on the start of the selected scan line for the transcoder attached to Pipe B.
	7	**Plane_3_B_Flip_Done_Event** This event is reported on the completion of a flip for Plane 3 B.
	6	**Plane_3_A_Flip_Done_Event** This event is reported on the completion of a flip for Plane 3 A.
	5	**Reserved**
	4	**Reserved**
	3	**Pipe_A_Start_of_Vertical_Blank_Event** This event is reported on the start of the vertical blank of the transcoder attached to Pipe A.
	2	**Plane_2_A_Flip_Done_Event** This event is reported on the completion of a flip for Plane 2 A.
	1	**Plane_1_A_Flip_Done_Event** This event is reported on the completion of a flip for Plane 1 A.
	0	**Pipe_A_Scanline_Event** This event is reported on the start of the selected scan line for the transcoder attached to Pipe A.

DstRegNum

DstRegNum	
Source:	Eulsa
Size (in bits):	8
Default Value:	0x00000000

Description

Register Number
The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number.
An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers.
RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero.
This field is present for the direct addressing mode and not present for indirect addressing.
This field applies to both source and destination operands.

DWord	Bit	Description
0	7:0	**Destination Register Number**

Value	Name	Description
0-127	If {Dst/Src0/Src1/Src2}.RegFile==GRF	
0-0ffh	If {Dst/Src0/Src1/Src2}.RegFile==ARF	This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

DstSubRegNum

DstSubRegNum

Source: Eulsa

Size (in bits): 5

Default Value: 0x00000000

Programming Notes

Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

DWord	Bit	Description
0	4:0	**Destination Sub Register Number** Value: 0-31, Name: If {Dst/Src0/Src1/Src2}.RegFile==GRF, Description: Value: 0-0ffh, Name: If {Dst/Src0/Src1/Src2}.RegFile==ARF, Description: This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

Dword Data Payload Register

MDCR_DW - Dword Data Payload Register

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Dword0** Format: U32 Specifies the slot 0 data in this payload register
0.1	31:0	**Dword1** Format: U32 Specifies the slot 1 data in this payload register
0.2	31:0	**Dword2** Format: U32 Specifies the slot 2 data in this payload register
0.3	31:0	**Dword3** Format: U32 Specifies the slot 3 data in this payload register
0.4	31:0	**Dword4** Format: U32 Specifies the slot 4 data in this payload register
0.5	31:0	**Dword5** Format: U32 Specifies the slot 5 data in this payload register
0.6	31:0	**Dword6** Format: U32 Specifies the slot 6 data in this payload register
0.7	31:0	**Dword7** Format: U32 Specifies the slot 7 data in this payload register

Dword SIMD4x2 Atomic CMPWR Message Data Payload

MDP_AOP4X2_DW2 - Dword SIMD4x2 Atomic CMPWR Message Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Src0 Slot0** Format: U32|S31|F32 Specifies the Slot 0 Source 0 data
1	31:0	**Src1 Slot0** Format: U32|S31|F32 Specifies the Slot 0 Source 1 data
2..3	63:0	**Reserved** Format: Ignore Ignored
4	31:0	**Src0 Slot1** Format: U32|S31|F32 Specifies the Slot 1 Source 0 data
5	31:0	**Src1 Slot1** Format: U32|S31|F32 Specifies the Slot 1 Source 1 data
6..7	63:0	**Reserved** Format: Ignore Ignored

Dword SIMD4x2 Atomic Operation Message Data Payload

MDP_AOP4X2_DW1 - Dword SIMD4x2 Atomic Operation Message Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Dword0** Format: U32|S31|F32 Specifies the Slot 0 Source or Return data
1..3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**Dword1** Format: U32|S31|F32 Specifies the Slot 1 Source or Return data
5..7	95:0	**Reserved** Format: Ignore Ignored

Dword SIMD4x2 Data Payload

MDP_DW_SIMD4X2 - Dword SIMD4x2 Data Payload

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Red Slot0** Format: U32 Specifies the Slot 0 red channel data
1	31:0	**Green Slot0** Format: U32 Specifies the Slot 0 green channel data
2	31:0	**Blue Slot0** Format: U32 Specifies the Slot 0 blue channel data
3	31:0	**Alpha Slot0** Format: U32 Specifies the Slot 0 alpha channel data
4	31:0	**Red Slot1** Format: U32 Specifies the Slot 1 red channel data
5	31:0	**Green Slot1** Format: U32 Specifies the Slot 1 green channel data
6	31:0	**Blue Slot1** Format: U32 Specifies the Slot 1 blue channel data
7	31:0	**Alpha Slot1** Format: U32 Specifies the Slot 1 alpha channel data

Dword SIMD8 Atomic Operation CMPWR Message Data Payload

MDP_AOP8_DW2 - Dword SIMD8 Atomic Operation CMPWR Message Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0** Format: MDCR_DW Specifies the Slot [7:0] Source 0 data
1.0-1.7	255:0	**Src1** Format: MDCR_DW Specifies the Slot [7:0] Source 1 data

Dword SIMD8 Data Payload

MDP_DW_SIMD8 - Dword SIMD8 Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Data[7:0]** Format: MDCR_DW Specifies the Slot [7:0] data

Dword SIMD16 Atomic Operation CMPWR Message Data Payload

MDP_AOP16_DW2 - Dword SIMD16 Atomic Operation CMPWR Message Data Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0[7:0]** Format: MDCR_DW Specifies the Source 0 data for Slot [7:0]
1.0-1.7	255:0	**Src0[15:8]** Format: MDCR_DW Specifies the Source 0 data for Slot [15:8]
2.0-2.7	255:0	**Src1[7:0]** Format: MDCR_DW Specifies the Source 1 data for Slot [7:0]
3.0-3.7	255:0	**Src1[15:8]** Format: MDCR_DW Specifies the Source 1 data for Slot [15:8]

Dword SIMD16 Data Payload

MDP_DW_SIMD16 - Dword SIMD16 Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[7:0]** Format: MDCR_DW Specifies the Slot [7:0] data
1.0-1.7	255:0	**Data[15:8]** Format: MDCR_DW Specifies the Slot [15:8] data

DX9_CONSTANTB_ENTRY

DX9_CONSTANTB_ENTRY		
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTB_* commands. Each entry provides the values for the one boolean constant being updated.		
DWord	**Bit**	**Description**
0	31:0	**Component** Format: U32 The boolean value to be stored.

DX9_CONSTANTF_ENTRY

DX9_CONSTANTF_ENTRY		
Source:	RenderCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTF_* commands. Each entry provides the values for the four components of one float constant being updated.		
DWord	**Bit**	**Description**
0	127:96	**Component 3** Format: IEEE_Float The 4th component of the nth float to be stored.
	95:64	**Component 2** Format: IEEE_Float The 3rd component of the nth float to be stored.
	63:32	**Component 1** Format: IEEE_Float The 2nd component of the nth float to be stored.
	31:0	**Component 0** Format: IEEE_Float The 1st component of the nth float to be stored.

DX9_CONSTANTI_ENTRY

DX9_CONSTANTI_ENTRY		
Source:	RenderCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTI_* commands. Each entry provides the values for the four components of one integer constant being updated.		
DWord	**Bit**	**Description**
0	31:0	**Component 0** Format: U32 The 1st component of the nth float to be stored.
1	31:0	**Component 1** Format: U32 The 2nd component of the nth float to be stored.
2	31:0	**Component 2** Format: U32 The 3rd component of the nth float to be stored.
3	31:0	**Component 3** Format: U32 The 4th component of the nth float to be stored.

Encoder Control State Parameters0

Encoder Control State Parameters0

Source: BSpec

Size (in bits): 320

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:30	**Reserved** Format: MBZ
	29	**Reserved**
	28:27	**AVC Encoder Chroma Sub-sample type** AVC YUV chroma compression mode. Value / Name: 00b: YUV 4:2:0 01b: RGBA 4:4:4:4 10b: YUV 4:4:4 11b: Reserved
	26:5	**Reserved** Format: MBZ
	4	**Reserved**
	3	**Reserved** Format: MBZ
	2	**Conditional Replenishment Enable** Format: U1 If the distortion for a MB is below a fixed threshold, it is coded as Pskip (No Coded Coeff). If above threshold, MB is coded as Intra type. **Programming Notes** This field must be set once per session. If this field is off, all frames are encoded as IDR frames, and frame number as 0. GOP Size parameter must be set to 1. This field should be turned off when IPCM is on.
	1	**Reserved** Format: MBZ
	0	**Reserved**
1	31:17	**Reserved** Format: MBZ

Encoder Control State Parameters0

	16	**GOP-level indirect user-defined NAL data packet - enable** GOP-level user defined NAL packet is inserted during the first frame of a GOP if this field is enabled. **Value** / **Name** / **Description** 0 / Disable / Disable insertion of indirect NAL data packet 1 / Enable / Enable insertion of indirect NAL data packet **Programming Notes** Recommended value is 0. Header contains several NAL units. AVC spec needs AUD NAL to be the first NAL for a frame. If GOP header and Frame headers are enabled and both have AUD NAL it would lead to incorrect bit stream. Multiple NAL data packets can be programmed within the GOP-level indirect surface. A linked list data structure is allocated multiple elements (NAL packets) in contiguous memory. Each element consists of link descriptor fields and datum. The first four bytes of each element contain the link descriptor fields: Byte 0: length_in_bytes[7:0] -- LSB length of data element/NAL packet. Byte 1: length_in_bytes[15:8] -- MSB length of data element/NAL packet. Byte 2: <{7'b000_0000, last_NAL} -- Bit 0 indicates the current element is the last element of the link list. Byte 3: Reserved -- Reserved- MBZ Datum starts at Byte 4 of each element: Byte 4: NAL_START Byte 5: ... Byte N: NAL_END Case 1: last_NAL =1 all the bytes through ByteN would be sent to BSP. Case 2: last_NAL =0 and ByteN is CL alighned (N mod 64 =0). Next NAL descriptor starts at N 1 Byte. Case 3: last_NAL = 0 and ByteN is not CL aligned zero bytes stuffed till next NAL descriptor (63 – (N mod 64) bytes of zeros). Note: empty/zero datum is not allowed
	15:0	**Reserved**
2	31:17	**Reserved** Format: MBZ
	16	**Frame-level indirect user-defined NAL data packet - enable** Frame-level user defined NAL packet is inserted during the first frame of a frame if this field is enabled. **Value** / **Name** / **Description** 0 / Disable / Disable insertion of indirect NAL data packet 1 / Enable / Enable insertion of indirect NAL data packet **Programming Notes**

Encoder Control State Parameters0

		The recommended value is 1. As mentioned in GOP header enable, enabling GOP header and frame header causes problem. Enabling only frame header is safe and sufficient.
		Multiple NAL data packets can be programmed within the frame-level indirect surface. A linked list data structure is allocated multiple elements (NAL packets) in contiguous memory. Each element consists of link descriptor fields and datum. The first four bytes of each element contain the link descriptor fields: Byte 0: length_in_bytes[7:0] -- LSB length of data element/NAL packet. Byte 1: length_in_bytes[15:8] -- MSB length of data element/NAL packet. Byte 2: <{7'b000_0000, last_NAL} -- Bit 0 indicates the current element is the last element of the link list. Byte 3: Reserved -- Reserved- MBZ Datum starts at Byte 4 of each element: Byte 4: NAL_START Byte 5: ... Byte N: NAL_END Note: empty/zero datum is not allowed
	15:0	**Reserved**
3	31:0	**Reserved** Format: MBZ
4	31:0	**Reserved** Format: MBZ
5	31:16	**GOP size parameter** Number of (P-Frames + 1) per GOP structure.

Value	Name	Description
0		Infinite number of P-frames followed by an initial I-frame(IDR)
1		GOP structure will contain one IDR frame. HME and IME inter predictions are disabled.
2		One IDR + one P-frame per GOP
3-FFFFh		This parameter indicates the number of frames within the GOP structure. Each GOP will have one IDR frame follows by the value of the parameter minus one P-frames.

	15:4	**Reserved** Format: MBZ
	3:0	**log2 Maximum Frame Number Minus4** This field specifies the value of the variable MaxFrameNum that is used in Frame_Num count for every frame and must match the definition of log2_max_frame_num_minus field in SPS NAL packet.

Value	Name
0-12	

Encoder Control State Parameters0

		Programming Notes The same value should be set for log2_max_pic_order_cnt_lsb_minus4 field in SPS NAL packet.
6	31:0	**Reserved** Format: MBZ
7	31	**Transform 8x8 Flag** This field indicates that 8x8 transform can be used within the frame. **Value** / **Name** / **Description** 0 / 4x4 Integer Transform / The MB must be set to 4x4 transform. 1 / 8x8 Integer Transform / The MB could be coded with 8x8 transform. **Programming Notes** When this field is set to 1, The transform_size_8x8_flag syntax element, if present in the output bit stream, is the same as this field; However, whether transform_szie_8x8_flag is present or not in the output bit stream depends on several other conditions. Hardware set MB level transform8x8 flag to 1 for two conditions: • It might be 1 if IntraMbFlag = INTRA and IntraMbMode = INTRA_8x8 • It must be 1 if IntraMbFlag = INTER and there is no sub partition size less than 8x8 Otherwise, this field must be set to 0. For Intra MB if transform8x8 = 0, hardware will always honor it, thus intra8x8 will be the winner. If transform8x8 = 0, Intra16x16 prediction Enable or Intra4x4 prediction Enable MUST be true.
	30:29	**Reserved** Format: MBZ
	28:24	**Second Chroma QP Offset, Chroma_qp_offset [9:5]** Format: S4 **Range:** -12 to +12 According to AVC Spec. It specifies the offset for determining QP Cr from QP Y. It is set to the upper 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS. • Chroma_qp_offset [9:5] - second_chroma_qp_offset_bits **Programming Notes** To ensure that the MB size doesn't exceed 3200 bits, Cr/Cb QP cannot go below 10. The Value of MinQp for Luma and Chroma Offset is programmed in such a way to ensure this. E.g. If chroma offset = -5, MinQp should be >= 15. This would ensure that the Final Chroma QP >= (-5+15 = 10).
	23:21	**Reserved** Format: MBZ

Encoder Control State Parameters0

	20:16	**Chroma QP Offset, Chroma_qp_offset[4:0]** Format: S4 **Range:** -12 to +12 According to AVC Spec. It specifies the offset for determining QP Cb from QP Y.It is set to the lower 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS. • Chroma_qp_offset [4:0] - chroma_qp_offset_bits (from the current active PPS) **Programming Notes** To ensure that the MB size doesn't exceed 3200 bits, Cr/Cb QP cannot go below 10. The Value of MinQp for Luma and Chroma Offset is programmed in such a way to ensure this. E.g. If chroma offset = -5, MinQp should be >= 15. This would ensure that the Final Chroma QP >= (-5+15 = 10).
	15:8	**Reserved** Format: MBZ
	7	**Round Inter Enable** Format: Enable **Programming Notes** Recommended driver setting is 0. When rounding Inter is disabled, a value of 2/8 is used for rounding inter coefficients.
	6:4	**Rounding Inter (N)** Format: **RoundingPrecisionTable_3_Bits** **Programming Notes** Hardware default this field to 2 if "Round Inter Enable" is disable.
	3	**Round Intra Enable** Format: Enable **Programming Notes** Recommended driver setting is 0. When rounding Intra is disabled, a value of 4/8 is used for rounding intra coefficients.
	2:0	**Rounding Intra (N)** Format: **RoundingPrecisionTable_3_Bits** **Programming Notes** Hardware default this field to 4 if "Round Intra Enable" is disable.

Encoder Control State Parameters0

8	31:0	**Reserved** Format: MBZ
9	31:19	**Reserved** Format: MBZ
	18:16	**Slice Pattern Per MB Row** The field should be set according to the following table and the MB row size. In cases where row cannot be divided evenly, round up to the nearest MB to achieve the indicated number of slices per row. For purposes of this clause, the rate of macroblocks/second is the rate that applies to the video being transmitted, according to the definition in the AVC standard, That is: the macroblock rate = ceiling(frame width, 16)/16 * ceiling(frame height, 16)/16 * (frames/second). Slice size Must be larger than 4MB.

Macroblocks/second Rate	Slice Pattern
rate < 250,000	1 row = 1 slice
250,000 <= rate < 500,000	1 row = 2 equal slices
500,000 <= rate < 1,000,000	1 row = 4 equal slices
1,000,000 <= rate < 2,100,000	1 row = 8 equal slices
2,100,000 <= rate	1 row = 16 equal slices

Value	Name	Description
0		1 slice per MB row
1		2 slice per MB row
2		4 slice per MB row
3		8 slice per MB row
4		16 slice per MB row

	15:0	**Reserved** Format: MBZ

Encoder Statistics Format

Encoder Statistics Format

Source: VideoEnhancementCS

Size (in bits): 128

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description

The per block data is intended for use by the video encoder and consists of 16 bytes of Denoise block data and FMD variances.
Much of the data is encoded as an 8-bit mantissa with the leading 1 removed and a 4-bit shift.
To recover the original 17-bit integer this code can be used:
If (exp != 0)
Number = ((0x100 | Mantissa) « exp) » 7;
else
Number = mantissa;

The values for STAD, SHCM and SVCM for each 4x4 are shited down by 2 bits to make 14-bit values before being summed for the 16x4 block to make a 16-bit value. The result is then converted into the mantissa/exp format.

DWord	Bit	Description
0	31:24	**Tearing_Count 1 (FMD Variance[8])** Format: U8 Number of pixels that have (diff_cTcB > diff_cTcT + diff_cBcB) Value / Name / Description: 0 / / DI is Disabled
	23:16	**Tearing_Count 2** Format: U8 If the frame is Deinterlaced with Top First in the DN/DI state then this is (FMD Variance[9]) = Number of pixels that have (diff_cTpB > diff_cTcT + diff_pBpB) If the frame is bottom first then this is (FMD Variance[10]) = Number of pixels that have (diff_cBpT > diff_pTpT + diff_cBcB) Value / Name / Description: 0 / / DI is Disabled
	15:8	**Motion_Count (FMD Variance[7])** Format: U8 Number of pixels that are moving (different above a threshold) Value / Name / Description: 0 / / DI is Disabled

Encoder Statistics Format

	7:0	**Reserved** Format: MBZ
1	31:28	**sSTAD** Format: U4 Shift for the Sum in time of absolute differences for 16x4. Value 0: Temporal Denoise Filtering is Disabled.
	27:24	**sSHCM** Format: U4 Shift for the Sum horizontaly of absolute differences. Value 0: DN is Disabled
	23:20	**sSVCM** Format: U4 Shift for the Sum vertically of absolute differences.
	19:16	**sDiff_cTpT** Format: U4 Shift for the sum of differences in top fields of current and previous frame. Value 0: DI is Disabled
	15:12	**sDiff_cBpB** Format: U4 Shift for the sum of differences in bottom field of current and previous frame. Value 0: DI is Disabled
	11:8	**sDiff_cTcB** Format: U4 Shift for the sum of differences between top and bottom field in current frame. Value 0: DI is Disabled
	7:4	**sDiff_cTpB** Format: U4 Shift for the sum of differences between current top and previous bottom. Value 0: DI is Disabled

Encoder Statistics Format

	3:0	**sDiff_cBpT** Format: U4 Shift for the sum of differences between current bottom and previous top. Value: 0 — Name: — Description: DI is Disabled
2	31:24	**mDiff_cBpB (FMD Variance[1])** Format: U8 Mantissa of sum of differences in bottom field of current and previous frame. Value: 0 — Name: — Description: DI is Disabled
	23:16	**mDiff_cTcB (FMD Variance[2])** Format: U8 Mantissa of sum of differences between top and bottom field in current frame. Value: 0 — Name: — Description: DI is Disabled
	15:8	**mDiff_cTpB (FMD Variance[3])** Format: U8 Mantissa of sum of differences between current top and previous bottom. Value: 0 — Name: — Description: DI is Disabled
	7:0	**mDiff_cBpT (FMD Variance[4])** Format: U8 Mantissa of sum of differences between current bottom and previous top. Value: 0 — Name: — Description: DI is Disabled
3	31:24	**mSTAD** Format: U8 Mantissa of Sum in time of absolute differences for 16x4. Value: 0 — Name: — Description: Temporal Denoise Filtering is disabled.
	23:16	**mSHCM** Format: U8 Mantissa of Sum horizontaly of absolute differences. Value: 0 — Name: — Description: DN is Disabled

Encoder Statistics Format

	15:8	**mSVCM** Format: U8 Mantissa of Sum vertically of absolute differences. **Value** / **Name** / **Description** 0 / / DN is Disabled
	7:0	**mDiff_cTpT (FMD Variance[0])** Format: U8 Mantissa of sum of differences in top fields of current and previous frame. **Value** / **Name** / **Description** 0 / / DI is Disabled

EU_INSTRUCTION_BASIC_ONE_SRC

EU_INSTRUCTION_BASIC_ONE_SRC		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

EU_INSTRUCTION_BASIC_THREE_SRC

EU_INSTRUCTION_BASIC_THREE_SRC

Source: EuIsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127	**Reserved** Format: MBZ
	126:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	105:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	84:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC
	63:56	**Destination Register Number** Format: DstRegNum
	55:53	**Destination Subregister Number**
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are "x", "y", "z", and "w", respectively, where "x" corresponds to Channel 0 in the group and "w" corresponds to channel 3 in the group
	48:46	**Destination Data Type** Value / Name / Description: 000b / :f / single precision Float (32-bit) 001b / :d / signed Doubleword integer 010b / :ud / Unsigned Doubleword integer 011b / :df / Double precision Float (64-bit) 100b / :hf / Half Float (16-bit) 101b-111b / Reserved /

EU_INSTRUCTION_BASIC_THREE_SRC

45:43 Source Data Type

Value	Name	Description
000b	:f	single precision Float (32-bit)
001b	:d	signed Doubleword integer
010b	:ud	Unsigned Doubleword integer
011b	:df	Double precision Float (64-bit)
100b	:hf	Half Float (16-bit)
101b-111b	Reserved	

42:41 Source 2 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	**SrcMod**

40:39 Source 1 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	**SrcMod**

42:37 Reserved

Exists If:	(Property[Source Modifier]=='false')
Format:	MBZ

38:37 Source 0 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	**SrcMod**

36 Source 1 Type

Format:	U1

Only used if Source Data Type is :f or :hf, else Source 1 Data Type matches Source 0 type and this bit is ignored.

Value	Name	Description
0b	:f	single precision Float (32-bit)
1b	:hf	Half Float (16-bit)

35 Source 2 Type

Format:	U1

Only used if Source Data Type is :f or :hf, else Source 2 Data Type matches Source 0 type and this bit is ignored.

Value	Name	Description
0b	:f	single precision Float (32-bit)
1b	:hf	Half Float (16-bit)

EU_INSTRUCTION_BASIC_THREE_SRC

	34	**MaskCtrl** (formerly WECtrl/Write Enable Control). This flag disables the normal write enables; it should normally be 0.

Value	Name	Description
0	Normal	Use the normal write enables in Dst.ChanEn (normal setting).
1	NoMask	Write all channels except those disabled by predication or by other masks besides the write enables.

Programming Notes
MaskCtrl = NoMask also skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

	33	**Flag Register Number** This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	32	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	31:0	**Header**

Format:	EU_INSTRUCTION_HEADER

EU_INSTRUCTION_BASIC_TWO_SRC

EU_INSTRUCTION_BASIC_TWO_SRC		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

EU_INSTRUCTION_BRANCH_CONDITIONAL

EU_INSTRUCTION_BRANCH_CONDITIONAL

Source: EuIsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127:64	**Sources** Exists If: ([Src1.RegFile]!='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_REG**
	127:64	**Sources** Exists If: ([Src1.RegFile]=='IMM') Format: **EU_INSTRUCTION_SOURCES_REG_IMM**
	63:48	**JIP** Format: S15 Jump Target Offset. The jump distance in number of eight-byte units if a jump is taken for the instruction.
	47	**Reserved** Format: MBZ
	46:44	**Src1.SrcType** Format: DataType This field specifies the numeric data type of the source operand src1. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on RegFile field of the source operand, there are two different encoding for this field. If a source is a register operand, this field follows the Source Register Type Encoding. If a source is an immediate operand, this field follows the Source Immediate Type Encoding. **Programming Notes** Both source operands, src0 and src1, support immediate types, but only one immediate is allowed for a given instruction and it must be the last operand. Halfbyte integer vector (v) type can only be used in instructions in packed-word execution mode. Therefore, in a two-source instruction where src1 is of type :v, src0 must be of type :b, :ub, :w, or :uw.
	43:42	**Src1.RegFile** Format: **RegFile**
	41:39	**Src0.SrcType** Format: DataType

EU_INSTRUCTION_BRANCH_CONDITIONAL

	38:37	**Src0.RegFile** Format: **RegFile**
	36:34	**Destination Data Type** Format: DataType This field specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send instruction, this field applies to the CurrDst ? the current destination operand.
	33:32	**Destination Register File** Format: **RegFile**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

EU_INSTRUCTION_BRANCH_ONE_SRC

EU_INSTRUCTION_BRANCH_ONE_SRC		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:96	**JIP** Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit**
	94:91	**Src1.SrcType** Format: **SrcType**
	90:89	**Src1.RegFile** Format: **RegFile**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	63:32	**Operand Control** Format: **EU_INSTRUCTION_OPERAND_CONTROLS**
	31:0	**Header** Format: **EU_INSTRUCTION_HEADER**

EU_INSTRUCTION_BRANCH_TWO_SRC

EU_INSTRUCTION_BRANCH_TWO_SRC		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:96	**JIP** Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:0	**Header** Format: EU_INSTRUCTION_HEADER

EU_INSTRUCTION_COMPACT_THREE_SRC

EU_INSTRUCTION_COMPACT_THREE_SRC		
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:57	**Src2.RegNum[6:0]** Format: SrcRegNum[6:0] Src2.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src2.RegNum[7]. Maps to 124:118
	56:50	**Src1.RegNum[6:0]** Format: SrcRegNum[6:0] Src1.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src1.RegNum[7]. Maps to 103:97
	49:43	**Src0.RegNum[6:0]** Format: SrcRegNum[6:0] Src0.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src0.RegNum[7]. Maps to 82:76
	42:40	**Src2.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 117:115
	39:37	**Src1.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 96:94
	36:34	**Src0.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 75:73

EU_INSTRUCTION_COMPACT_THREE_SRC

	33	**Src2.RepCtrl** Format: **RepCtrl** Maps to 106
	32	**Src1.RepCtrl** Format: **RepCtrl** Maps to 85
	31	**Reserved** Exists If: (Property[Saturation]=='false') Format: MBZ
	31	**Saturate** Exists If: (Property[Saturation]=='true') Maps to 31
	30	**DebugCtrl** Format: DebugCtrl
	29	**Compaction Control** Format: CmptCtrl
	28	**Src0.RepCtrl** Format: **RepCtrl** Maps to 64
	27:19	**Reserved** Format: MBZ
	18:12	**Dst.RegNum[6:0]** Format: **DstRegNum[6:0]** Dst.RegNum[7:0] with MSB of zero and [6:0] from the compact instruction Maps to 63:56 (Dst.RegNum)

EU_INSTRUCTION_COMPACT_THREE_SRC

	11:10	**SourceIndex** Lookup one of four 46-bit values. That value is used (from MSB to LSB) for the Src2.RegNum[7], Src1.RegNum[7], Src0.RegNum[7], Src2.ChanSel, Src1.ChanSel, Src0.ChanSel, Dst.SubRegNum, Dst.ChanEnable, Dst.DstType, SrcType, Src2.Modifier, Src1.Modifier, and Src0.Modifier bit fields. Maps to 125, 104, 83, 114:107, 93:86, 72:65, 55:49, 48:43, 42:37

Value	Name	Description
0	00011100100111001001110010000011110000000000000	No Negation
1	00011100100111001001110010000011110000000000010	Negate Src0
2	00011100100111001001110010000011110000000001000	Negate Src1
3	00011100100111001001110010000011110000000100000	Negate Src2

	9:8	**ControlIndex** Lookup one of four 24-bit values. That value is used (from MSB to LSB) for the MaskCtrl, FlagRegNum/FlagSubRegNum, AccWrCtrl, CondModifier, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, NibCtrl, DepCtrl, and AccessMode bit fields. Maps to 34, 33:32, 28:8

Value	Name	Description
0	100000000110000000000001	(8) Q1 NoMask Align16
1	000000000110000000000001	(8) Q1 Align16
2	000000001000000000000001	(16) H1 Align16
3	000000001000000000100001	(16) H2 Align16

	7	**Reserved** Format: MBZ
	6:0	**Opcode**

EU_INSTRUCTION_COMPACT_TWO_SRC

EU_INSTRUCTION_COMPACT_TWO_SRC

Source:	EuIsa
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

The following table describes the EU compact instruction format. The compact instruction format for 1 or 2-source instructions is essentially identical to the compact instruction format for earlier generations, but the compact fields expand to somewhat different fields in the native instruction format, as the native instruction format changed.

DWord	Bit	Description
0..1	63:56	**Src1.RegNum** Exists If: ([DataTypeIndex][Src1.RegFile]!='IMM') Format: **SrcRegNum** Maps to 108:101 (Src1.RegNum)
	63:56	**Src1.RegNum** Exists If: ([DataTypeIndex][Src1.RegFile]=='IMM') Maps to 103:96 (Imm32[7:0])
	55:48	**Src0.RegNum** Format: **SrcRegNum** Maps to 76:69 (Src0.RegNum)
	47:40	**Dst.RegNum** Format: **DstRegNum** Maps to 60:53 (Dst.RegNum)
	39:35	**Src1Index** Exists If: ([DataTypeIndex][Src1.RegFile]!='IMM') Format: **SrcIndex** If not an immediate operand, lookup one of 32 12-bit values that maps to bits 120:109. That value is used (from MSB to LSB) for the Src1.VertStride, various Src1 bit fields based on AccessMode (Src1.ChanSel[7:4], Src1.Width, Src1.HorzStride), Src1.AddrMode, and Src1.SrcMod bit fields Maps to 120:109

EU_INSTRUCTION_COMPACT_TWO_SRC

39:35 Src1Index

Exists If:	([DataTypeIndex][Src1.RegFile]=='IMM')

If an immediate operand, there is no lookup. Determines bits 127:104 (Imm32[31:8]) as follows: map bits 39:35 directly to bits 108:104. Sign extend to fill bits 127:109. Compact format bit 39 is thus copied to all of bits 127:108 for an immediate operand.

Maps to 127:104

34:30 Src0Index

Format:	SrcIndex

Lookup one of 32 12-bit values. That value is used (from MSB to LSB) for the Src0.VertStride, various Src0 bit fields based on AccessMode (Src0.ChanSel[7:4], Src0.Width, Src0.HorzStride), Src0.AddrMode, and Src0.SrcMod bit fields. Note that this field spans a DWord boundary within the QWord compacted instruction.

Maps to 88:77

29 Compaction Control

Format:	CmptCtrl

28 Reserved

Format:	MBZ

27:24 Reserved

Exists If:	(Property[Conditional Modifier]=='false')
Format:	MBZ

27:24 Conditional Modifier

Exists If:	(Property[Conditional Modifier]=='true')
Format:	CondModifier

23 Accumulator Write Control

Format:	AccWrCtrl

22:18 SubRegIndex

Lookup one of 32 15-bit values. That value is used (from MSB to LSB) for various fields for Src1, Src0, and Dst, including ChanEn/ChanSel, SubRegNum, and AddrImm[4] or AddrImm[4:0], depending on AddrMode and AccessMode.

Maps to 100:96, 68:64, 52:48

Value	Name	Description
0	000000000000000	0 | 0 | 0 |
1	000000000000001	0.x | 0.xx | 0.xx
2	000000000001000	8 | 0 | 0 |
3	000000000001111	0.xyzw | 0.xx | 0.xx

EU_INSTRUCTION_COMPACT_TWO_SRC

		4	000000000010000	16 | 0 | 0 |
		5	000000010000000	0 | 4 | 0 |
		6	000000100000000	0 | 8 | 0 |
		7	000000110000000	0 | 12 | 0 |
		8	000001000000000	0 | 16 | 0 |
		9	000001000010000	16 | 16 | 0 |
		10	000001010000000	0 | 20 | 0 |
		11	001000000000000	0 | 0 | 4 |
		12	001000000000001	0.x | 0.xx | 0.xy
		13	001000010000001	0.x | 0.xy | 0.xy
		14	001000010000010	0.y | 0.xy | 0.xy
		15	001000010000011	0.xy | 0.xy | 0.xy
		16	001000010000100	0.z | 0.xy | 0.xy
		17	001000010000111	0.xyz | 0.xy | 0.xy
		18	001000010001000	0.w | 0.xy | 0.xy
		19	001000010001110	0.yzw | 0.xy | 0.xy
		20	001000010001111	0.xyzw | 0.xy | 0.xy
		21	001000110000000	0 | 12 | 4 |
		22	001000111101000	0.w | 0.ww | 0.xy
		23	010000000000000	0 | 0 | 8 |
		24	010000110000000	0 | 12 | 8 |
		25	011000000000000	0 | 0 | 12 |
		26	011110010000111	0.xyz | 0.xy | 0.ww
		27	100000000000000	0 | 0 | 16 |
		28	101000000000000	0 | 0 | 20 |
		29	110000000000000	0 | 0 | 24 |
		30	111000000000000	0 | 0 | 28 |
		31	111000000011100	28 | 0 | 28 |

17:13	**DataTypeIndex**

Lookup one of 32 21-bit values. That value is used (from MSB to LSB) for the Dst.AddrMode, Dst.HorzStride, Src1.SrcType, Src1.RegFile, Src0.SrcType, Src0.RegFile, Dst.DstType, and Dst.RegFile bit fields.

Maps to 63:61, 94:89, 46:35

Value	Name	Description
0	00100000000000000001	r:ud | a:ud | a:ud | <1> | dir |

EU_INSTRUCTION_COMPACT_TWO_SRC

		1	00100000000001000000	a:ud | r:ud | a:ud | <1> | dir |
		2	00100000000001000001	r:ud | r:ud | a:ud | <1> | dir |
		3	00100000000011000001	r:ud | i:ud | a:ud | <1> | dir |
		4	00100000000101011101	r:f | r:d | a:ud | <1> | dir |
		5	00100000010111011101	r:f | i:vf | a:ud | <1> | dir |
		6	00100000011101000001	r:ud | r:f | a:ud | <1> | dir |
		7	00100000011101000101	r:d | r:f | a:ud | <1> | dir |
		8	00100000011101011101	r:f | r:f | a:ud | <1> | dir |
		9	00100001000001000001	r:ud | r:ud | r:ud | <1> | dir |
		10	00100001100000001000000	a:ud | r:ud | i:ud | <1> | dir |
		11	00100001100000001000001	r:ud | r:ud | i:ud | <1> | dir |
		12	00100010100010100101	r:d | r:d | r:d | <1> | dir |
		13	00100011100010100100	a:d | r:d | i:d | <1> | dir |
		14	00100011100010100101	r:d | r:d | i:d | <1> | dir |
		15	00101110001110101101	r:f | r:f | a:f | <1> | dir |
		16	00101110101110001101	r:f | a:f | r:f | <1> | dir |
		17	00101110101110101100	a:f | r:f | r:f | <1> | dir |
		18	00101110101110101101	r:f | r:f | r:f | <1> | dir |
		19	00101111101110101100	a:f | r:f | i:f | <1> | dir |
		20	00000000001000001100	a:w | a:ub | a:ud | <0> | dir |
		21	00100000000001011101	r:f | r:ud | a:ud | <1> | dir |
		22	00100000000101000101	r:d | r:d | a:ud | <1> | dir |
		23	00100001000001000000	a:ud | r:ud | r:ud | <1> | dir |
		24	00100010100010100100	a:d | r:d | r:d | <1> | dir |
		25	00100011100010000100	a:d | a:d | i:d | <1> | dir |
		26	00100100100100001001	r:uw | a:uw | r:uw | <1> | dir |
		27	00101011101110101101	r:f | r:f | i:vf | <1> | dir |
		28	00101111101110101101	r:f | r:f | i:f | <1> | dir |
		29	00100111100110100100	a:w | r:w | i:w | <1> | dir |
		30	00100100100100100100	a:uw | r:uw | r:uw | <1> | dir |
		31	00100101100100100100	a:uw | r:uw | i:uw | <1> | dir |

EU_INSTRUCTION_COMPACT_TWO_SRC

12:8 **ControlIndex**

Lookup one of 32 19-bit values. That value is used (from MSB to LSB) for the FlagRegNum, FlagSubRegNum, Saturate, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, DepCtrl, MaskCtrl, and AccessMode bit fields.

Maps to 33:32, 31, 23:12, 10:9, 34, 8

Value	Name	Description
0	0000000000000000010	Align1 | We | (1) | f0.0
1	0000100000000000000	Align1 | (4) | f0.0
2	0000100000000000001	Align16 | (4) | f0.0
3	0000100000000000010	Align1 | We | (4) | f0.0
4	0000100000000000011	Align16 | We | (4) | f0.0
5	0000100000000000100	Align1 | NoDDClr | (4) | f0.0
6	0000100000000000101	Align16 | NoDDClr | (4) | f0.0
7	0000100000000000111	Align16 | We | NoDDClr | (4) | f0.0
8	0000100000000001000	Align1 | NoDDChk | (4) | f0.0
9	0000100000000001001	Align16 | NoDDChk | (4) | f0.0
10	0000100000000001101	Align16 | NoDDClr, NoDDChk | (4) | f0.0
11	0000110000000000000	Align1 | Q1 | (8) | f0.0
12	0000110000000000001	Align16 | Q1 | (8) | f0.0
13	0000110000000000010	Align1 | We | Q1 | (8) | f0.0
14	0000110000000000011	Align16 | We | Q1 | (8) | f0.0
15	0000110000000000100	Align1 | NoDDClr | Q1 | (8) | f0.0
16	0000110000000000101	Align16 | NoDDClr | Q1 | (8) | f0.0
17	0000110000000000111	Align16 | We | NoDDClr | Q1 | (8) | f0.0
18	0000110000000001001	Align16 | NoDDChk | Q1 | (8) | f0.0
19	0000110000000001101	Align16 | NoDDClr, NoDDChk | Q1 | (8) | f0.0
20	0000110000000010000	Align1 | Q2 | (8) | f0.0
21	0000110000100000000	Align1 | Q1 | +f.xyzw | (8) | f0.0
22	0001000000000000000	Align1 | H1 | (16) | f0.0
23	0001000000000000010	Align1 | We | H1 | (16) | f0.0
24	0001000000000000100	Align1 | NoDDClr | H1 | (16) | f0.0
25	0001000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.0
26	0010110000000000000	Align1 | Q1 | (8) | .sat | f0.0
27	0010110000000010000	Align1 | Q2 | (8) | .sat | f0.0
28	0011000000000000000	Align1 | H1 | (16) | .sat | f0.0
29	0011000000100000000	Align1 | H1 | +f.xyzw | (16) | .sat | f0.0
30	0101000000000000000	Align1 | H1 | (16) | f0.1
31	0101000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.1

EU_INSTRUCTION_COMPACT_TWO_SRC		
	7	**DebugCtrl** Format: DebugCtrl
	6:0	**Opcode**

EU_INSTRUCTION_CONTROLS_A

EU_INSTRUCTION_CONTROLS_A

Source: EuIsa
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:13	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	12	**PredInv** Exists If: (Property[Predication]=='true') This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields.
		Value / **Name** / **Description** 0 / Positive **[Default]** / Positive polarity of predication. Use the predication mask produced by PredCtrl 1 / Negative / Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
	12	**Reserved** Exists If: (Property[Predication]=='false')
	11:8	**PredCtrl** Exists If: (Property[Predication]=='true') Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register. Encoding depends on the access mode. In Align16 access mode, there are eight encodings (including no predication). All encodings are based on group-of-4 predicate bits, including channel sequential, replication swizzles and horizontal any|all operations. The same configuration is repeated for each group-of-4 execution channels.
	11:8	**Reserved** Exists If: (Property[Predication]=='false') Format: **PredCtrl**

EU_INSTRUCTION_CONTROLS_A

	7:6	**Thread Control** Format: **ThreadCtrl** Thread Control. This field provides explicit control for thread switching. If this field is set to 00b, it is up to the GEN execution units to manage thread switching. This is the normal (and unnamed) mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. In another example, if the current instruction is ready to go, however, there is another thread with higher priority that also has an instruction ready, the EU switches to that thread. If this field is set to Switch, a forced thread switch occurs after the current instruction is executed and before the next instruction. In addition, a long delay (longer than the execution pipe latency) is introduced for the current thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. Switch is designed primarily as a safety feature in case there are race conditions for certain instructions.
	5:4	**QtrCtrl** Format: **QtrCtrl** **Quarter Control.** This field provides explicit control for ARF selection. This field combined with NibCtrl and ExecSize determines which channels are used for the ARF registers.
	3	**NibCtrl** Nibble Control. This field is used in some instructions along with QtrCtrl. See the description of QtrCtrl below. NibCtrl is only used for SIMD4 instructions with a DF (Double Float) source or destination.

Value	Name	Description
0	Odd	Use an odd 1/8th for DMask/VMask and ARF (first, third, fifth, or seventh depending on QtrCtrl).
1	Even	Use an even 1/8th for DMask/VMask and ARF (second, fourth, sixth, or eighth depending on QtrCtrl).

Programming Notes
Note that if eighths are given zero-based indices from 0 to 7, then NibCtrl = 0 indicates even indices and NibCtrl = 1 indicates odd indices.

EU_INSTRUCTION_CONTROLS_A

	2:1	**DepCtrl** Format: DepCtrl Destination Dependency Control. This field selectively disables destination dependency check and clear for this instruction. When it is set to 00, normal destination dependency control is performed for the instruction - hardware checks for destination hazards to ensure data integrity. Specifically, destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard will be cleared when the destination operands retire. When bit 10 is set (NoDDClr), the destination register scoreboard will NOT be cleared when the destination operands retire. When bit 11 is set (NoDDChk), hardware does not check for destination register dependency before the instruction is made ready for execution. NoDDClr and NoDDChk are not mutual exclusive. When this field is not all-zero, hardware does not protect against destination hazards for the instruction. This is typically used to assemble data in a fine grained fashion (e.g. matrix-vector compute with dot-product instructions), where the data integrity is guaranteed by software based on the intended usage of instruction sequences.
	0	**AccessMode** Access Mode. This field determines the operand access for the instruction. It applies to all source and destination operands. When it is cleared (Align1), the instruction uses byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are not supported. When it is set (Align16), the instruction uses 16-byte-aligned addressing for all source and destination operands. Source swizzle control and destination mask control are supported in this mode.

Value	Name
0	Align1 **[Default]**
1	Align16

EU_INSTRUCTION_CONTROLS_B

EU_INSTRUCTION_CONTROLS_B

Source: Eulsa
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3	**Saturate** Exists If: (Property[Saturation]=='true') Enables or disables destination saturation. When it is set, output values to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any value outside the saturation target range for the data type to the closest value in the target range. For a floating-point destination type, the saturation target range is [0.0, 1.0]. For a floating-point NaN, there is no *closest value*; any NaN saturates to 0.0. Note that enabling Saturate overrides all of the NaN propagation behaviors described for various numeric instructions. Any floating-point number greater than 1.0, including +INF, saturates to 1.0. Any negative floating-point number, including -INF, saturates to 0.0. Any floating-point number in the range 0.0 to 1.0 is not changed by saturation. For an integer destination type, the maximum range for that type is the saturation target range. For example, the saturation range for B (Signed Byte Integer) is [-128, 127]. When Saturate is clear, destination values are not saturated. For example, a wrapped result (modulo) is output to the destination for an overflowed integer value. See the Numeric Data Typessection for information about data types and their ranges. Value / Name / Description 0 / No destination modification **[Default]** / 1 / sat / Saturate the output
	3	**Reserved** Exists If: (Property[Saturation]=='false') Format: MBZ
	2	**DebugCtrl** This field allows the insertion of a breakpoint at the current instruction. When the bit is set, hardware automatically stores the current IP in CR register and jumps to the System IP (SIP) BEFORE executing the current instruction. Value / Name 0 / No Breakpoint **[Default]** 1 / Breakpoint

EU_INSTRUCTION_CONTROLS_B

	1	**CmptCtrl** Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

	0	**AccWrCtrl** AccWrCtrl. This field allows per instruction accumulator write control.

Value	Name	Description
0	Don't write to ACC **[Default]**	
1	Update ACC	Write result to the ACC, and destination

EU_INSTRUCTION_CONTROLS

EU_INSTRUCTION_CONTROLS		
Source:	Eulsa	
Size (in bits):	24	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	23:20	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	19:16	**Reserved** Exists If: (Property[Conditional Modifier]=='false') Format: MBZ
	19:16	**CondModifier** Exists If: (Property[Conditional Modifier]=='true') Format: CondModifier Does not exist for send/sendc/math/branch/break-continue opcodes
	15:0	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A

EU_INSTRUCTION_HEADER

EU_INSTRUCTION_HEADER		
Source:	Eulsa	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:8	**Control** Format: EU_INSTRUCTION_CONTROLS
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

EU_INSTRUCTION_ILLEGAL

EU_INSTRUCTION_ILLEGAL		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

EU_INSTRUCTION_MATH

EU_INSTRUCTION_MATH		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Format: EU_INSTRUCTION_SOURCES_REG_REG
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Function Control (FC)** Format: FC
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

EU_INSTRUCTION_NOP

EU_INSTRUCTION_NOP		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:31	**Reserved** Format: MBZ
	30	**DebugCtrl** This field allows the insertion of a breakpoint at the current instruction. When the bit is set, hardware automatically stores the current IP in CR register and jumps to the System IP (SIP) BEFORE executing the current instruction. **Value** / **Name** 0 — No Breakpoint **[Default]** 1 — Breakpoint
	29:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

EU_INSTRUCTION_OPERAND_CONTROLS

EU_INSTRUCTION_OPERAND_CONTROLS

Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Destination Register Region** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_DST_ALIGN16**
	31:16	**Destination Register Region** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_DST_ALIGN1**
	15	**Reserved** Exists If: ([Destination Register Region][Destination Addressing Mode]=='Direct') Format: MBZ
	15	**Destination Address Immediate[9:9]** Exists If: ([Destination Register Region][Destination Addressing Mode]=='Indirect') Format: U1
	14:11	**Src0.SrcType** Exists If: ([Src0.RegFile]!='IMM') Format: **SrcType**
	14:11	**Src0.SrcType** Exists If: ([Src0.RegFile]=='IMM') Format: **SrcImmType**
	10:9	**Src0.RegFile** Format: **RegFile**
	8:5	**Destination Data Type** Format: **DstType** This field specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send instruction, this field applies to the CurrDst - the current destination operand.
	4:3	**Destination Register File** Format: **RegFile**

EU_INSTRUCTION_OPERAND_CONTROLS

	2	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0". **Value** / **Name** / **Description** 0 / Normal **[Default]** / 1 / Write all channels / Except channels killed with predication control **Programming Notes** MaskCtrl = NoMask skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.
	1:0	**Flag Register Number/Subregister Number**

EU_INSTRUCTION_OPERAND_DST_ALIGN1

EU_INSTRUCTION_OPERAND_DST_ALIGN1

Source: Eulsa
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15	**Destination Addressing Mode** Format: **AddrMode** For a send instruction, this field applies to PostDst - the post destination operand. Addressing mode for CurrDst (current destination operand) is fixed as Direct. (See Instruction Reference chapter for CurrDst and PostDst.)
	14:13	**Destination Horizontal Stride** Format: **HorzStride** For a send instruction, this field applies to CurrDst. PostDst only uses the register number.
	12:9	**Destination Address Subregister Number** Exists If: ([Destination Addressing Mode]=='Indirect') Format: **AddrSubRegNum** For a send instruction, this field applies to PostDst
	12:5	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: **DstRegNum** For a send instruction, this field applies to PostDst.
	8:0	**Destination Address Immediate** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S8 For a send instruction, this field applies to PostDst.
	4:0	**Destination Subregister Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: **DstSubRegNum** For a send instruction, this field applies to CurrDst.

EU_INSTRUCTION_OPERAND_DST_ALIGN16

EU_INSTRUCTION_OPERAND_DST_ALIGN16		
Source:	Eulsa	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Destination Addressing Mode** Format: **AddrMode** For a send instruction, this field applies to PostDst - the post destination operand. Addressing mode for CurrDst (current destination operand) is fixed as Direct. (See Instruction Reference chapter for CurrDst and PostDst.)
	14:13	**Reserved** Format: MBZ
	12:9	**Destination Address Subregister Number** Exists If: ([Destination Addressing Mode]=='Indirect') Format: **AddrSubRegNum** For a send instruction, this field applies to PostDst
	12:5	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: **DstRegNum** For a send instruction, this field applies to PostDst.
	8:4	**Destination Address Immediate[8:4]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S8[8:4] For a send instruction, this field applies to PostDst
	4	**Destination Subregister Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: **DstSubRegNum[4:4]** For a send instruction, this field applies to CurrDst.
	3:0	**Destination Channel Enable** Format: **ChanEn[4]** For a send instruction, this field applies to the CurrDst

EU_INSTRUCTION_OPERAND_SEND_MSG

EU_INSTRUCTION_OPERAND_SEND_MSG

Source: EuIsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**EOT** **Description** This field controls the termination of the thread. For a send instruction, if this field is set, EU will terminate the thread and also set the EOT bit in the message sideband. This field only applies to the send instruction. It is not present for other instructions. **Value** / **Name** 0 — Thread is not terminated 1 — EOT
	30:0	**Message Descriptor** Exists If: [SelReg32Desc]=='IMM' Format: **MsgDescpt31**
	30:0	**Reg32** Exists If: [SelReg32Desc]!='IMM' In a send or sendc instruction refers to the option of providing the message descriptor field DWord, of which bits 30:0 are used, in the first two words of the Address Register rather than as an immediate operand.

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1

Source: EuIsa
Size (in bits): 25
Default Value: 0x00000000

DWord	Bit	Description
0	24:21	**Source Vertical Stride** Format: **VertStride**
	20:18	**Source Width** Format: **Width**
	17:16	**Source Horizontal Stride** Format: **HorzStride**
	15	**Source Addressing Mode** Format: **AddrMode**
	14:13	**Source Modifier** Exists If: (Property[Source Modifier]=='true') Format: **SrcMod**
	14:13	**Reserved** Exists If: (Property[Source Modifier]=='false') Format: MBZ
	12:9	**Source Address Subregister Number** Exists If: ([Source Addressing Mode]=='Indirect') Format: **AddrSubRegNum**
	12:5	**Source Register Number** Exists If: ([Source Addressing Mode]=='Direct') Format: **SrcRegNum**
	8:0	**Source Address Immediate [8:0]** Exists If: ([Source Addressing Mode]=='Indirect') Format: S9[8:0]
	4:0	**Source Subregister Number** Exists If: ([Source Addressing Mode]=='Direct') Format: **SrcSubRegNum**

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

Source: Eulsa
Size (in bits): 25
Default Value: 0x00000000

DWord	Bit	Description
0	24:21	**Source Vertical Stride** Format: **VertStride**
	20	**Reserved** Format: MBZ
	19:16	**Source Channel Select[7:4]** Format: **ChanSel[4][7:4]**
	15	**Source Addressing Mode** Format: **AddrMode**
	14:13	**Source Modifier** Exists If: (Property[Source Modifier]=='true') Format: **SrcMod**
	14:13	**Reserved** Exists If: (Property[Source Modifier]=='false') Format: MBZ
	12:9	**Source Address Subregister Number** Exists If: ([Source Addressing Mode]=='Indirect') Format: **AddrSubRegNum**
	12:5	**Source Register Number** Exists If: ([Source Addressing Mode]=='Direct') Format: **SrcRegNum**
	8:4	**Source Address Immediate[8:4]** Exists If: ([Source Addressing Mode]=='Indirect') Format: S9[8:4]
	4	**Source Subregister Number[4:4]** Exists If: ([Source Addressing Mode]=='Direct') Format: **SrcSubRegNum[4:4]**
	3:0	**Source Channel Select[3:0]** Format: **ChanSel[4][3:0]**

EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC		
Source:	Eulsa	
Size (in bits):	21	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	20	**Source Subregister Number [1]** Format: SrcSubRegNum[1]
	19:12	**Source Register Number** Format: SrcRegNum
	11:9	**Source Subregister Number [4:2]** Format: SrcSubRegNum[4:2]
	8:1	**Source Swizzle** Format: ChanSel[4]
	0	**Source Replicate Control** Format: RepCtrl

EU_INSTRUCTION_SEND

EU_INSTRUCTION_SEND		
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95	**Reserved** Format: MBZ
	94:91	**ExDesc[31:28]** Format: ExtMsgDescpt[31:28]
	90:89	**Reserved** Format: MBZ
	88:85	**ExDesc[27:24]** Format: ExtMsgDescpt[27:24]
	84	**Reserved** Format: MBZ
	83:80	**ExDesc[23:20]** Format: ExtMsgDescpt[23:20]
	79:68	**Reserved** Format: MBZ
	67:64	**ExDesc[19:16]** Format: ExMsgDescpt[19:16]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B
	27:24	**Shared Function ID (SFID)** Format: SFID
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A
	7	**Reserved** Format: MBZ

EU_INSTRUCTION_SEND		
	6:0	**Opcode** Format: EU_OPCODE

EU_INSTRUCTION_SENDS

EU_INSTRUCTION_SENDS

Source: Eulsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG
	95:80	**ExDesc[31:16]** Format: ExtMsgDescpt[31:16]
	79	**Source 0 Addressing Mode** Format: AddrMode
	78	**Reserved** Exists If: ([Source 0 Addressing Mode]=='Direct') Format: MBZ
	78	**Source 0 Address Immediate Sign [9]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[9]
	77	**SelReg32Desc**
	76:73	**Source 0 Address Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Indirect')
	76:69	**Source 0 Register Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	72:68	**Source 0 Address Immediate [8:4]** Exists If: ([Source 0 Addressing Mode]=='Indirect') Format: S9[8:4]
	68	**Source 0 Subregister Number** Exists If: ([Source 0 Addressing Mode]=='Direct')
	67:64	**ExDesc[9:6]** Format: ExtMsgDescpt[9:6]
	63	**Destination Addressing Mode** Format: AddrMode
	62	**Destination Address Immediate Sign [9]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[9]

EU_INSTRUCTION_SENDS

	62	**Reserved** Exists If: ([Destination Addressing Mode]=='Direct') Format: MBZ
	61	**Reserved** Format: MBZ
	60:57	**Destination Address Subregister Number** Exists If: ([Destination Addressing Mode]=='Indirect')
	60:53	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct')
	56:52	**Destination Address Immediate [8:4]** Exists If: ([Destination Addressing Mode]=='Indirect') Format: S9[8:4]
	52	**Destination Subregister Number [4]** Exists If: ([Destination Addressing Mode]=='Direct')
	51:44	**Source 1 Register Number**
	43:41	**Reserved** Format: MBZ
	40:37	**Destination Type**
	36	**Source 1 Register File** Format: **RegFile[0]**
	35	**Destination Register File** Format: **RegFile[0]**
	34	**MaskCtrl**
	33:32	**Flag Register Number/Subregister Number**
	31:28	**Controls B** Format: **EU_INSTRUCTION_CONTROLS_B**
	27:24	**Shared Function ID (SFID)** Format: **SFID**
	23:8	**Controls A** Format: **EU_INSTRUCTION_CONTROLS_A**
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: **EU_OPCODE**

EU_INSTRUCTION_SOURCES_IMM32

EU_INSTRUCTION_SOURCES_IMM32		
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Single source, immediate		
DWord	**Bit**	**Description**
0..1	63:32	**Source 0 Immediate**
	31:25	**Reserved** Format: MBZ
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**

EU_INSTRUCTION_SOURCES_REG

EU_INSTRUCTION_SOURCES_REG		
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Single source, register		
DWord	**Bit**	**Description**
0..1	63:25	**Reserved** Format: MBZ
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**

EU_INSTRUCTION_SOURCES_REG_IMM

EU_INSTRUCTION_SOURCES_REG_IMM		
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Dual source, register and immediate		
DWord	**Bit**	**Description**
0..1	63:32	**Source 1 Immediate**
	31	**Reserved** Exists If: ([Source 0][Source Addressing Mode]=='Direct') Format: MBZ
	31	**Source 0 Address Immediate [9] (Sign Bit)** Exists If: ([Source 0][Source Addressing Mode]=='Indirect') Format: S9[9]
	30:27	**Src1.SrcType** Format: **SrcImmType**
	26:25	**Src1.RegFile** Format: **RegFile**
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**

EU_INSTRUCTION_SOURCES_REG_REG

EU_INSTRUCTION_SOURCES_REG_REG		
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Dual source, both registers		
DWord	**Bit**	**Description**
0..1	63:58	**Reserved** Format: MBZ
	57	**Reserved** Exists If: ([Source 1][Source Addressing Mode]=='Direct') Format: MBZ
	57	**Source 1 Address Immediate [9] (Sign Bit)** Exists If: ([Source 1][Source Addressing Mode]=='Indirect') Format: S9[9]
	56:32	**Source 1** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16**
	56:32	**Source 1** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: **EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1**
	31	**Reserved** Exists If: ([Source 0][Source Addressing Mode]=='Direct') Format: MBZ
	31	**Source 0 Address Immediate [9] (Sign Bit)** Exists If: ([Source 0][Source Addressing Mode]=='Indirect') Format: S9[9]

EU_INSTRUCTION_SOURCES_REG_REG

30:27 **Src1.SrcType**

Format:	SrcType

This field specifies the numeric data type of the source operand src1. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on RegFile field of the source operand, there are two different encoding for this field. If a source is a register operand, this field follows the Source Register Type Encoding. If a source is an immediate operand, this field follows the Source Immediate Type Encoding.

Value	Name
11b	Reserved

Programming Notes
Both source operands, src0 and src1, support immediate types, but only one immediate is allowed for a given instruction and it must be the last operand.
Halfbyte integer vector (v) type can only be used in instructions in packed-word execution mode. Therefore, in a two-source instruction where src1 is of type :v, src0 must be of type :b, :ub, :w, or :uw.

26:25 **Src1.RegFile**

Format:	RegFile

24:0 **Source 0**

Exists If:	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM')
Format:	EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

24:0 **Source 0**

Exists If:	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM')
Format:	EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1

Execution_Unit_Extended_Message_Descriptor

Execution_Unit_Extended_Message_Descriptor

Source: BSpec
Size (in bits): 11
Default Value: 0x00000000

DWord	Bit	Description
0	10	**Reserved** Format: MBZ
	9:6	**Reserved** Exists If: //(Structure[EU_INSTRUCTION_HEADER][Opcode]=='Send' OR Structure[EU_INSTRUCTION_HEADER][Opcode]=='Sendc') Format: MBZ
	9:6	**Extended Message Length** Exists If: //(Structure[EU_INSTRUCTION_HEADER][Opcode]=='Sends' OR Structure[EU_INSTRUCTION_HEADER][Opcode]=='Sendsc') Format: U4 This field specifies the number of 256-bit GRF registers starting from <src1> to be sent out on the request message payload. Valid value ranges from 0 to 15. Must be 0 when <src1> is null register. Value: [0,15] — Name:
	5	**End Of Thread** This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed. This bit maps to bit 127 of the send/sends instruction. 0 — No Termination 1 — EOT
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** This field indicates the function unit for which the message is intended. **Refer to "GPU Overview" document for the mapping of Shared Function IDs**

Extended Message Descriptor - Execution Unit

Extended Message Descriptor - Execution Unit

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Extended Function Control** Exists If: //(Structure[EU_INSTRUCTION_HEADER][Opcode]!='Sel') OR ([SelReg32ExDesc]=='IMM') Format: U16
	31:12	**Extended Function Control** Exists If: //(Structure[EU_INSTRUCTION_HEADER][Opcode]=='Sel') AND ([SelReg32ExDesc]!='IMM') Format: U20 This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
	15:12	**Reserved** Exists If: //(Structure[EU_INSTRUCTION_HEADER][Opcode]!='Sel') OR ([SelReg32ExDesc]=='IMM') Format: MBZ
	11	**Reserved** Format: MBZ
	10:0	**Execution Unit Extended Message Descriptor Definition** Format: Execution_Unit_Extended_Message_Descriptor

Extended Message Descriptor Render Target

Extended Message Descriptor Render Target		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:20	**Reserved** Format: MBZ
	15:12	**Reserved** Format: MBZ
	9:6	**Extended Message Length** Format: U4 This field specifies the number of 256-bit GRF registers starting from <src1> to be sent out on the request message payload. Valid value ranges from 0 to 15. Must be 0 when <src1> is null register.
	5	**End of Thread** This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** This field indicates the function unit for which the message is intended. *Refer to "GPU Overview" document for the mapping of Shared Function IDs*

Extended Message Descriptor - Sampling Engine

Extended Message Descriptor - Sampling Engine		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:12	**Bindless Surface Offset** Format: BindlessSurfaceOffset[25:6] Specifies the bindless surface offset if the **Binding Table Index** is set to 252. Ignored otherwise. The bindless surface offset is added to the **Bindless Surface Base Address** as bits 25:6 of the byte-based address. The resulting address is the location of SURFACE_STATE for this message.
	11	**Reserved** Format: MBZ
	10:0	**Execution Unit Extended Message Descriptor Definition** Format: Execution_Unit_Extended_Message_Descriptor

ExtMsgDescpt

ExtMsgDescpt

Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0 Extended Message Descriptor Definition for SendS (Immediate)	31:16	**Extended Function Control** Format: U16 This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
	15:12	**Reserved** Format: MBZ
	11	**Reserved** Format: MBZ
	10:6	**Reserved** Format: MBZ
	5	**EOT** Format: U1 This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed. Value: 0 — Name: No Termination Value: 1 — Name: EOT
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** Format: U4 If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Value: 0000b — Name: Null Value: 0001b — Name: Reserved Value: 0010b — Name: SamplingEngine Value: 0011b — Name: MessageGateway

ExtMsgDescpt			
		0100b	DataCacheDataPort2
		0101b	DataPortRenderCache
		0110b	URB
		0111b	ThreadSpawner
		1000b	VideoMotionEstimation
		1001b	DataCacheReadOnlyDataPort
		1010b	DataCacheDataPort
		1011b	PixelInterpolator
		1100b	DataCacheDataPort 1
		1101b	CheckandRefinementEngine
		[1110b,1111b]	Reserved

ExtMsgDescptImmediate

ExtMsgDescptImmediate

Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0 Extended Message Descriptor Definition for SendS (Immediate)	31:16	**Extended Function Control** Format: U16 This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
	15:12	**Reserved** Format: MBZ
	11	**Reserved** Format: MBZ
	10	**Reserved** Format: MBZ
	9:6	**Reserved** Format: MBZ
	5	**EOT** Format: U1 This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed. Value: 0 — Name: No Termination Value: 1 — Name: EOT
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** Format: U4 If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Value: 0000b — Name: Null Value: 0001b — Name: Reserved

ExtMsgDescptImmediate			
		0010b	SamplingEngine
		0011b	MessageGateway
		0100b	DataCacheDataPort2
		0101b	DataPortRenderCache
		0110b	URB
		0111b	ThreadSpawner
		1000b	VideoMotionEstimation
		1001b	DataCacheReadOnlyDataPort
		1010b	DataCacheDataPort
		1011b	PixelInterpolator
		1100b	DataCacheDataPort 1
		1101b	CheckandRefinementEngine
		[1110b,1111b]	Reserved

FFTID Message Header Control

MHC_FFTID - FFTID Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:8	**Reserved** Format: Ignore Ignored
	7:0	**FFTID** Format: U8 Fixed function thread ID, used to free up resources by the thread on thread completion.

Filter_Coefficient

Filter_Coefficient		
Source:	BSpec	
Size (in bits):	8	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	7:0	**Filter Coefficient** Format: S1.6 2's Complement Range : [-1 63/64, +1 63/64]

Filter_Coefficients

Filter_Coefficients		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	63:56	**Filter Coefficient Offset 7** Format: **Filter_Coefficient**
	55:48	**Filter Coefficient Offset 6** Format: **Filter_Coefficient**
	47:40	**Filter Coefficient Offset 5** Format: **Filter_Coefficient**
	39:32	**Filter Coefficient Offset 4** Format: **Filter_Coefficient**
	31:24	**Filter Coefficient Offset 3** Format: **Filter_Coefficient**
	23:16	**Filter Coefficient Offset 2** Format: **Filter_Coefficient**
	15:8	**Filter Coefficient Offset 1** Format: **Filter_Coefficient**
	7:0	**Filter Coefficient Offset 0** Format: **Filter_Coefficient**

FrameDeltaQp

FrameDeltaQp		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:56	**FrameDeltaQp[7]** Format: S7
	55:48	**FrameDeltaQp[6]** Format: S7
	47:40	**FrameDeltaQp[5]** Format: S7
	39:32	**FrameDeltaQp[4]** Format: S7
	31:24	**FrameDeltaQp[3]** Format: S7
	23:16	**FrameDeltaQp[2]** Format: S7
	15:8	**FrameDeltaQp[1]** Format: S7
	7:0	**FrameDeltaQp[0]** Format: S7

FrameDeltaQpRange

FrameDeltaQpRange		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:56	**FrameDeltaQpRange[7]** Format: U8
	55:48	**FrameDeltaQpRange[6]** Format: U8
	47:40	**FrameDeltaQpRange[5]** Format: U8
	39:32	**FrameDeltaQpRange[4]** Format: U8
	31:24	**FrameDeltaQpRange[3]** Format: U8
	23:16	**FrameDeltaQpRange[2]** Format: U8
	15:8	**FrameDeltaQpRange[1]** Format: U8
	7:0	**FrameDeltaQpRange[0]** Format: U8

FunctionControl

FunctionControl		
Source:	Eulsa	
Size (in bits):	6	
Default Value:	0x00000000	

DWord	Bit	Description
0	5:4	**Reserved**
	3:0	**Target Function ID**

Value	Name
0000b	Reserved
0001b	INV (Reciprocal)
0010b	LOG
0011b	EXP
0100b	SQRT
0101b	RSQ
0110b	SIN
0111b	COS
1000b	Reserved
1001b	FDIV
1010b	POW
1011b	INT DIV Quotient and remainder
1100b	INT DIV Quotient only
1101b	INT DIV Remainder only
1110b	INVM
1111b	RSQRTM

GATHER_CONSTANT_ENTRY

GATHER_CONSTANT_ENTRY		
Source:	RenderCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:8	**Constant Buffer Offset** Format: Offset[7:0]ConstantBuffer This field specifies the Offset in 128-bit units of the 128b entry fetched from the constant buffer for this entry (including when **On-Die Table Read Enable** is set).
	7:4	**Channel Mask** Mask: Mask[3:0] Format: ConstantBuffer Each bit of this field correspond to the 4 channels of each entry fetched from memory. When the bit is a 1, the corresponding 32-bit value is loaded in FF's push constant buffer. When the bit is a 0, the corresponding 32-bit value is not loaded. If this field is zero it means the entry is not used.
	3:0	**Binding Table Index Offset** Format: Constant Buffer Index offset [3:0]Surface State for ConstantBuffer This field specifies the Binding Table index offset from the **Constant Buffer Binding Table Block** starting point in the Binding Table. This value is added to the **Constant Buffer Binding Table Block** will result in the Binding Table Index pointing to the surface state containing the constant buffer to be referenced. If **VS Constant Buffer Dx9 Enable** is set then a value of '1' specifies that the fetch to the constant buffer should be offset by 4KB in order to address the upper 4K of the constant buffer. Any value greater than '1' is invalid when **VS Constant Buffer Dx9 Enable** is set.

GraphicsAddress63-0

GA63-0 - GraphicsAddress63-0		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. For the specified product generations, use 48-bit addressing.	63:48	**Reserved** Format: MBZ
	47:0	**GraphicsAddress47-0** Format: GraphicsAddress[47:0]

GraphicsAddress63-1

GA63-1 - GraphicsAddress63-1

Source:	BSpec
Size (in bits):	63
Default Value:	0x00000000, 0x00000000

This structure is intended to define the upper bits of the GraphicsAddress, when bit 0 is already defined in the referring register. So bit 0 of this structure should correspond to bit 1 of the full GraphicsAddress.

DWord	Bit	Description
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	62:47	**Reserved** Format: MBZ
	46:0	**GraphicsAddress47-1** Format: GraphicsAddress[47:1] Bits 47:1 of a 48-bit GraphicsAddress. Look for bit 0 definition in the referring register.

GraphicsAddress63-2

GA63-2 - GraphicsAddress63-2

Source:	BSpec
Size (in bits):	62
Default Value:	0x00000000, 0x00000000

This structure is intended to define the upper bits of the GraphicsAddress, when bits 1:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 2 of the full GraphicsAddress.

DWord	Bit	Description
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	61:46	**Reserved** Format: MBZ
	45:0	**GraphicsAddress47-2** Format: GraphicsAddress[47:2] Bits 47:2 of a 48-bit GraphicsAddress. Look for the definition of bits 1:0 in the referring register.

GraphicsAddress63-3

GA63-3 - GraphicsAddress63-3		
Source:	BSpec	
Size (in bits):	61	
Default Value:	0x00000000, 0x00000000	
This structure is intended to define the upper bits of the GraphicsAddress, when bits 2:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 3 of the full GraphicsAddress.		
DWord	**Bit**	**Description**
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	60:45	**Reserved** Format: MBZ
	44:0	**GraphicsAddress47-3** Format: GraphicsAddress[47:3] Bits 47:2 of a 48-bit GraphicsAddress. Look for the definition of bits 2:0 in the referring register.

GraphicsAddress63-4

GA63-4 - GraphicsAddress63-4

Source:	BSpec
Size (in bits):	60
Default Value:	0x00000000, 0x00000000

This structure is intended to define the upper bits of the GraphicsAddress, when bits 3:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 4 of the full GraphicsAddress.

DWord	Bit	Description
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	59:44	**Reserved** Format: MBZ
	43:0	**GraphicsAddress47-4** Format: GraphicsAddress[47:4] Bits 47:4 of a 48-bit GraphicsAddress. Look for the definition of bits 3:0 in the referring register.

GraphicsAddress63-5

GA63-5 - GraphicsAddress63-5		
Source:	BSpec	
Size (in bits):	59	
Default Value:	0x00000000, 0x00000000	
This structure is intended to define the upper bits of the GraphicsAddress, when bits 4:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 5 of the full GraphicsAddress.		
DWord	**Bit**	**Description**
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	58:43	**Reserved** Format: MBZ
	42:0	**GraphicsAddress47-5** Format: GraphicsAddress[47:5] Bits 47:5 of a 48-bit GraphicsAddress. Look for the definition of bits 4:0 in the referring register.

GraphicsAddress63-6

GA63-6 - GraphicsAddress63-6

Source:	BSpec
Size (in bits):	58
Default Value:	0x00000000, 0x00000000

This structure is intended to define the upper bits of the GraphicsAddress, when bits 5:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 6 of the full GraphicsAddress.

DWord	Bit	Description
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	57:42	**Reserved** Format: MBZ
	41:0	**GraphicsAddress47-6** Format: GraphicsAddress[47:6] Bits 47:6 of a 48-bit GraphicsAddress. Look for the definition of bits 5:0 in the referring register.

GraphicsAddress63-12

GA63-12 - GraphicsAddress63-12		
Source:	BSpec	
Size (in bits):	52	
Default Value:	0x00000000, 0x00000000	
This structure is intended to define the upper bits of the GraphicsAddress, when bits 11:0 are already defined in the referring register. So bit 0 of this structure should correspond to bit 12 of the full GraphicsAddress.		
DWord	**Bit**	**Description**
0..1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and must be zero. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.	51:36	**Reserved** Format: MBZ
	35:0	**GraphicsAddress47-12** Format: GraphicsAddress[47:12] Bits 47:12 of a 48-bit GraphicsAddress. Look for the definition of bits 11:0 in the referring register.

GTC Interrupt Bit Definition

GTC Interrupt Bit Definition		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
The GTC Interrupt Registers all share the same bit definitions from this table.		
DWord	**Bit**	**Description**
0	31:26	**Reserved** Format: MBZ
	25	**GTC Lock Loss** GTC has lost lock with a remote GTC sink. The difference between the local and remote GTC has exceeded programmed threshold.
	24	**GTC Aux Rx Error portA** An aux channel error occurred during GTC transfer with remote GTC sink attached to this port.
	23	**GTC Update Complete portA** A hardware initiated GTC update has completed with a sink attached to this port.
	22	**Reserved** Format: MBZ
	21:20	**Reserved**
	19:18	**Reserved**
	17	**GTC Aux Rx Error portD** An aux channel error occured during GTC transfer with remote GTC sink attached to this port.
	16	**GTC Update Complete portD** A hardware initiated GTC update has completed with a sink attached to this port.
	15:10	**Reserved** Format: MBZ
	9	**GTC Aux Rx Error portC** An aux channel error occurred during GTC transfer with remote GTC sink attached to this port.
	8	**GTC Update Complete portC** A hardware initiated GTC update has completed with a sink attached to this port.
	7:2	**Reserved** Format: MBZ
	1	**GTC Aux Rx Error portB** An aux channel error occurred during GTC transfer with remote GTC sink attached to this port.
	0	**GTC Update Complete portB** A hardware initiated GTC update has completed with a sink attached to this port.

Half Precision Dual Source SIMD8 Message Data Payload Register

MDPR_DSH_SIMD8 - Half Precision Dual Source SIMD8 Message Data Payload Register

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**Src0 Data1** Format: F16 Specifies the source 0 slot 1 data in this payload register
	15:0	**Src0 Data0** Format: F16 Specifies the source 0 slot 0 data in this payload register
1	31:16	**Src0 Data3** Format: F16 Specifies the source 0 slot 3 data in this payload register
	15:0	**Src0 Data2** Format: F16 Specifies the source 0 slot 2 data in this payload register
2	31:16	**Src0 Data5** Format: F16 Specifies the source 0 slot 5 data in this payload register
	15:0	**Src0 Data4** Format: F16 Specifies the source 0 slot 4 data in this payload register
3	31:16	**Src0 Data7** Format: F16 Specifies the source 0 slot 7 data in this payload register

MDPR_DSH_SIMD8 - Half Precision Dual Source SIMD8 Message Data Payload Register

	15:0	**Src0 Data6** Format: F16 Specifies the source 0 slot 6 data in this payload register
4	31:16	**Src1 Data1** Format: F16 Specifies the source 1 slot 1 data in this payload register
	15:0	**Src1 Data0** Format: F16 Specifies the source 1 slot 0 data in this payload register
5	31:16	**Src1 Data3** Format: F16 Specifies the source 1 slot 3 data in this payload register
	15:0	**Src1 Data2** Format: F16 Specifies the source 1 slot 2 data in this payload register
6	31:16	**Src1 Data5** Format: F16 Specifies the source 1 slot 5 data in this payload register
	15:0	**Src1 Data4** Format: F16 Specifies the source 1 slot 4 data in this payload register
7	31:16	**Src1 Data7** Format: F16 Specifies the source 1 slot 7 data in this payload register
	15:0	**Src1 Data6** Format: F16 Specifies the source 1 slot 6 data in this payload register

Half Precision OM Replicated SIMD16 Render Target Data Payload

MDP_RTWH_M16REP - Half Precision OM Replicated SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK Slots [15:0] oMask
1.0-1.7	255:0	**RGBA** Format: MDPR_H_RGBA RGBA for all slots [15:0]

Half Precision OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_MA8 - Half Precision OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision OM S0A SIMD16 Render Target Data Payload

MDP_RTWH_MA16 - Half Precision OM S0A SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Source 0 Alpha
1.0-1.7	255:0	**oMask** Format: MDPR_OMASK Slots [15:0] oMask
2.0-2.7	255:0	**Red[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Red
3.0-3.7	255:0	**Green[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Green
4.0-4.7	255:0	**Blue[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Blue
5.0-5.7	255:0	**Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Alpha

Half Precision OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_M8DS - Half Precision OM SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK oMask for slots [7:0] and [15:8]. Operation selects upper or lower half.
1.0-1.7	255:0	**Red** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Red
2.0-2.7	255:0	**Green** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Green
3.0-3.7	255:0	**Blue** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Blue
4.0-4.7	255:0	**Alpha** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Alpha

Half Precision OM SIMD8 Render Target Data Payload

MDP_RTWH_M8 - Half Precision OM SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK Slots [7:0] oMask. Upper half ignored.
1.0-1.7	255:0	**Red** Format: MDPR_H_SIMD8 Slots [7:0] Red
2.0-2.7	255:0	**Green** Format: MDPR_H_SIMD8 Slots [7:0] Green
3.0-3.7	255:0	**Blue** Format: MDPR_H_SIMD8 Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Alpha

Half Precision OM SIMD16 Render Target Data Payload

MDP_RTWH_M16 - Half Precision OM SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK Slots [15:0] oMask
1.0-1.7	255:0	**Red[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Red
2.0-2.7	255:0	**Green[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Green
3.0-3.7	255:0	**Blue[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Blue
4.0-4.7	255:0	**Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Alpha

Half Precision OS OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_SMA8 - Half Precision OS OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1792
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_SMA8 - Half Precision OS OM S0A SIMD8 Render Target Data Payload

6.0-6.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

Half Precision OS OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SM8DS - Half Precision OS OM SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision OS OM SIMD8 Render Target Data Payload

MDP_RTWH_SM8 - Half Precision OS OM SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision OS S0A SIMD8 Render Target Data Payload

MDP_RTWH_SA8 - Half Precision OS S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision OS SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_S8DS - Half Precision OS SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Red
1.0-1.7	255:0	**Green** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Green
2.0-2.7	255:0	**Blue** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Alpha
4.0-4.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] or [15:8] of Stencil

Half Precision OS SIMD8 Render Target Data Payload

MDP_RTWH_S8 - Half Precision OS SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_H_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDPR_H_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDPR_H_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Alpha
4.0-4.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

Half Precision OS SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_SZMA8 - Half Precision OS SZ OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_SZMA8 - Half Precision OS SZ OM S0A SIMD8 Render Target Data Payload

7.0-7.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

Half Precision OS SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SZM8DS - Half Precision OS SZ OM SIMD8 Dual Source Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

Half Precision OS SZ OM SIMD8 Render Target Data Payload

MDP_RTWH_SZM8 - Half Precision OS SZ OM SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

Half Precision OS SZ S0A SIMD8 Render Target Data Payload

MDP_RTWH_SZA8 - Half Precision OS SZ S0A SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

Half Precision OS SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SZ8DS - Half Precision OS SZ SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision OS SZ SIMD8 Render Target Data Payload

MDP_RTWH_SZ8 - Half Precision OS SZ SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision Replicated Pixel Render Target Data Payload Register

MDPR_H_RGBA - Half Precision Replicated Pixel Render Target Data Payload Register

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**Green** Format: U16 Specifies the value of all slots' green channel.
	15:0	**Red** Format: U16 Specifies the value of all slots' red channel.
1	31:16	**Alpha** Format: U16 Specifies the value of all slots' alpha channel.
	15:0	**Blue** Format: U16 Specifies the value of all slots' blue channel.
2..7	191:0	**Reserved** Format: Ignore Ignored

Half Precision Replicated SIMD16 Render Target Data Payload

MDP_RTWH_16REP - Half Precision Replicated SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**RGBA** Format: MDPR_H_RGBA RGBA for all slots [15:0]

Half Precision S0A SIMD8 Render Target Data Payload

MDP_RTWH_A8 - Half Precision S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Red** Format: MDPR_H_SIMD8 Slots [7:0] Red
2.0-2.7	255:0	**Green** Format: MDPR_H_SIMD8 Slots [7:0] Green
3.0-3.7	255:0	**Blue** Format: MDPR_H_SIMD8 Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Alpha

Half Precision S0A SIMD16 Render Target Data Payload

MDP_RTWH_A16 - Half Precision S0A SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Source 0 Alpha
1.0-1.7	255:0	**Red[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Red
2.0-2.7	255:0	**Green[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Green
3.0-3.7	255:0	**Blue[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Blue
4.0-4.7	255:0	**Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Alpha

Half Precision SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_8DS - Half Precision SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Red
1.0-1.7	255:0	**Green** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Green
2.0-2.7	255:0	**Blue** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Alpha

Half Precision SIMD8 Message Data Payload Register

MDPR_H_SIMD8 - Half Precision SIMD8 Message Data Payload Register

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**Data1** Format: F16 Specifies the slot 1 data in this payload register
	15:0	**Data0** Format: F16 Specifies the slot 0 data in this payload register
1	31:16	**Data3** Format: F16 Specifies the slot 3 data in this payload register
	15:0	**Data2** Format: F16 Specifies the slot 2 data in this payload register
2	31:16	**Data5** Format: F16 Specifies the slot 5 data in this payload register
	15:0	**Data4** Format: F16 Specifies the slot 4 data in this payload register
3	31:16	**Data7** Format: F16 Specifies the slot 7 data in this payload register
	15:0	**Data6** Format: F16 Specifies the slot 6 data in this payload register
4..7	127:0	**Reserved** Format: Ignore Ignored

Half Precision SIMD8 Render Target Data Payload

MDP_RTWH_8 - Half Precision SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_H_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDPR_H_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDPR_H_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Alpha

Half Precision SIMD16 Message Data Payload Register

MDPR_H_SIMD16 - Half Precision SIMD16 Message Data Payload Register

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**Data1** Format: F16 Specifies the slot 1 data in this payload register
	15:0	**Data0** Format: F16 Specifies the slot 0 data in this payload register
1	31:16	**Data3** Format: F16 Specifies the slot 3 data in this payload register
	15:0	**Data2** Format: F16 Specifies the slot 2 data in this payload register
2	31:16	**Data5** Format: F16 Specifies the slot 5 data in this payload register
	15:0	**Data4** Format: F16 Specifies the slot 4 data in this payload register
3	31:16	**Data7** Format: F16 Specifies the slot 7 data in this payload register
	15:0	**Data6** Format: F16 Specifies the slot 6 data in this payload register

MDPR_H_SIMD16 - Half Precision SIMD16 Message Data Payload Register

4	31:16	**Data9** Format: F16 Specifies the slot 9 data in this payload register
	15:0	**Data8** Format: F16 Specifies the slot 8 data in this payload register
5	31:16	**Data11** Format: F16 Specifies the slot 11 data in this payload register
	15:0	**Data10** Format: F16 Specifies the slot 10 data in this payload register
6	31:16	**Data13** Format: F16 Specifies the slot 13 data in this payload register
	15:0	**Data12** Format: F16 Specifies the slot 12 data in this payload register
7	31:16	**Data15** Format: F16 Specifies the slot 15 data in this payload register
	15:0	**Data14** Format: F16 Specifies the slot 14 data in this payload register

Half Precision SIMD16 Render Target Data Payload

MDP_RTWH_16 - Half Precision SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Red
1.0-1.7	255:0	**Green[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Green
2.0-2.7	255:0	**Blue[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Blue
3.0-3.7	255:0	**Alpha[15:0]** Format: MDPR_H_SIMD16 Slots [15:0] Alpha

Half Precision SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_ZMA8 - Half Precision SZ OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1792
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_ZMA8 - Half Precision SZ OM S0A SIMD8 Render Target Data Payload

6.0-6.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth

Half Precision SZ OM S0A SIMD16 Render Target Data Payload

MDP_RTWH_ZMA16 - Half Precision SZ OM S0A SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_ZMA16 - Half Precision SZ OM S0A SIMD16 Render Target Data Payload

7.0-7.7	255:0	**Source Depth[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Source Depth

Half Precision SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_ZM8DS - Half Precision SZ OM SIMD8 Dual Source Render Target Data Payload

Source: BSpec

Size (in bits): 1536

Default Value: 0x00000000, 0x00000000, 0x00000000,

Half Precision SZ OM SIMD8 Render Target Data Payload

MDP_RTWH_ZM8 - Half Precision SZ OM SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision SZ OM SIMD16 Render Target Data Payload

MDP_RTWH_ZM16 - Half Precision SZ OM SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_ZM16 - Half Precision SZ OM SIMD16 Render Target Data Payload		
6.0-6.7	255:0	**Source Depth[15:8]** Format: **MDP_DW_SIMD8** Slots [15:8] Source Depth

Half Precision SZ S0A SIMD8 Render Target Data Payload

MDP_RTWH_ZA8 - Half Precision SZ S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Half Precision SZ S0A SIMD16 Render Target Data Payload

MDP_RTWH_ZA16 - Half Precision SZ S0A SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1792
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTWH_ZA16 - Half Precision SZ S0A SIMD16 Render Target Data Payload		
6.0-6.7	255:0	**Source Depth[15:8]** Format: **MDP_DW_SIMD8** Slots [15:8] Source Depth

Half Precision SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_Z8DS - Half Precision SZ SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Red
1.0-1.7	255:0	**Green** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Green
2.0-2.7	255:0	**Blue** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_DSH_SIMD8 Slots[7:0] or [15:8] of Src0 and Src1 Alpha
4.0-4.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] or [15:8] of Source Depth

Half Precision SZ SIMD8 Render Target Data Payload

MDP_RTWH_Z8 - Half Precision SZ SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDPR_H_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDPR_H_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDPR_H_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDPR_H_SIMD8 Slots [7:0] Alpha
4.0-4.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth

Half Precision SZ SIMD16 Render Target Data Payload

MDP_RTWH_Z16 - Half Precision SZ SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

Hardware-Detected Error Bit Definitions

Hardware-Detected Error Bit Definitions

Source: RenderCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:8	**Reserved** Format: MBZ
	7	**Reserved**
	6:3	**Reserved**
	2	**Command Privilege Violation Error** This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: **Value** / **Name** / **Description** 1 / / Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

Hardware Status Page Layout

Hardware Status Page Layout	
Source:	RenderCS
Size (in bits):	32768
Default Value:	0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout		
1..3	31:0	**Reserved** Must not be used.
4	31:0	**Ring Head Pointer Storage** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
5..15	31:0	**Reserved** Must not be used.
16..27	31:0	**Context Status DWords**
28..30	31:0	**Reserved** Must not be used.
31	31:0	**Last Written Status Offset**
32..39	31:0	**Reserved**
40..46	31:0	**Reserved**
47	31:0	**Reserved**
48..1023	31:0	**General Purpose** These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

HCP_PAK_INSERT_OBJECT_INDIRECT_PAYLOAD

HCP_PAK_INSERT_OBJECT_INDIRECT_PAYLOAD		
Source:	VideoCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Indirect Payload Data Size in bits** Format: U32 Number of bits to be inserted. Not including those skipped bytes in the beginning. For VP9: the Data is always valid from start of cache-line, no offset is allowed.
1..2	63:0	**Indirect Payload Base Address** Format: SplitBaseAddress64ByteAligned 48-bit address of the indirect payload data in memory buffer. **Programming Notes** Payload must begin in a byte position, but the payload can be ended in a bit position.
3	31:0	**Indirect Payload Base Address** Format: MemoryAddressAttributes

Header Forbidden Message Descriptor Control Field

MDC_MHF - Header Forbidden Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Format: Enumeration Indicates the message forbids a message header. Value / Name / Description 0h / No **[Default]** / Message header is not present 1h / Reserved / Not used

Header Present Message Descriptor Control Field

MDC_MHP - Header Present Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Format: Enumeration Specifies if the message uses the optional message header. Value / Name / Description: 0h / No / Message header is not present 1h / Yes / Message header is present

Header Required Message Descriptor Control Field

MDC_MHR - Header Required Message Descriptor Control Field

Source:	BSpec
Size (in bits):	1
Default Value:	0x00000001

DWord	Bit	Description
0	0	**Message Header Present** Format: Enumeration Indicates the message requires a message header. 0h – Reserved – Not used 1h – Yes **[Default]** – Message header is present

HEVC_ARBITRATION_PRIORITY

HEVC_ARBITRATION_PRIORITY

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

DWord	Bit	Description
0	1:0	**Priority** Format: U2 Value / Name: 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority

HW Generated BINDING_TABLE_STATE

HW Generated BINDING_TABLE_STATE

Source:	BSpec
Size (in bits):	16
Default Value:	0x00000000

Description

The binding table binds surfaces to logical resource indices used by shaders and other compute engine kernels. The HW generated Binding_Table_State have different format than the SW generated Binding_Table_State. The HW generated Binding_Table_State is stored as an array of 256 elements, each of which contains one word as defined here. The start of each element is spaced one word apart. The first element of the binding table is aligned to a 64-byte boundary. Binding table indexes beyond 256 will automatically be mapped to entry 0 by the HW, w/ the exception of any messages which support the special indexes 240 through 255, inclusive.

DWord	Bit	Description
0	15:0	**Surface State Pointer** Format: SurfaceStateOffset[21:6]

Hword 1 Block Data Payload

MDP_HW1 - Hword 1 Block Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Hword** Format: U256 Specifies the Hword data

Hword 2 Block Data Payload

MDP_HW2 - Hword 2 Block Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Hword0** Format: U256 Specifies the Hword data for element 0
1.0-1.7	255:0	**Hword1** Format: U256 Specifies the Hword data for element 1

Hword 4 Block Data Payload

MDP_HW4 - Hword 4 Block Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Hword0** Format: U256 Specifies the Hword data for element 0
1.0-1.7	255:0	**Hword1** Format: U256 Specifies the Hword data for element 1
2.0-2.7	255:0	**Hword2** Format: U256 Specifies the Hword data for element 2
3.0-3.7	255:0	**Hword3** Format: U256 Specifies the Hword data for element 3

Hword 8 Block Data Payload

MDP_HW8 - Hword 8 Block Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_HW8 - Hword 8 Block Data Payload		
6.0-6.7	255:0	**Hword6** Format: U256 Specifies the Hword data for element 6
7.0-7.7	255:0	**Hword7** Format: U256 Specifies the Hword data for element 7

Hword Channel Mode Message Header Control

MHC_A64_CMODE - Hword Channel Mode Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Reserved** Format: MDC_CMODE Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	30:0	**Reserved** Format: Ignore Ignored

Hword Register Blocks Message Descriptor Control Field

MDC_DB_HW - Hword Register Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**Register Blocks** Format: Enumeration Specifies the number of Hword blocks to be read or written

Value	Name	Description
00h	HW1	1 Hword register
01h	HW2	2 Hword registers
02h	HW4	4 Hword registers
03h	HW8	8 Hword registers

Ignored Message Header

MH_IGNORE - Ignored Message Header		
Source:	DataPort 0	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
Some messages require a message header or have an optional message header, but do not use any information in the header.		
DWord	**Bit**	**Description**
0..7	255:0	**Reserved** Format: Ignore Ignored

Inline Data Description for MFD_AVC_BSD_Object

Inline Data Description for MFD_AVC_BSD_Object

Source: VideoCS

Size (in bits): 96

Default Value: 0x00000000, 0x00000000, 0x00000000

This structure includes all the required Slice Header parameters and error handling settings for AVC_BSD_OBJECT Command (DW3..DW5).

DWord	Bit	Description
0	31	**Concealment Method** This field specifies the method used for concealment when error is detected. If set, a copy from collocated macroblock location is performed from the concealment reference indicated by the ConCeal_Pic_Id field. If it is not set, a copy from the current picture is performed using Intra 16x16 Prediction method. Value / Name / Description: 0 — Intra 16x16 Prediction 1 — Inter P Copy
	30	**Init Current MB Number** When set, the current Slice_Start_MB_Num, Slice_MB_Start_Hor_Pos and Slice_MB_Start_Vert_Pos fields will be used to initialize the Current_MB_Number register. This effectively disables the concealment capability.
	29	**Intra PredMode (4x4/8x8 Luma) Error Control Bit** This field controls if AVC decoder will fix Intra Prediction Mode if the decoded value is incorrect according to MB position Value / Name / Description: 0 — AVC decoder will detect and fix IntraPredMode (4x4/8x8 Luma) Errors. 1 — AVC decoder will NOT detect IntraPredMode (4x4/8x8 Luma) Errors. The wrong IntraPredMode value will be retaind.
	28:27	**MB Error Concealment B Temporal Prediction mode** These two bits control how the reference L0/L1 are overridden in B temporal slice. Value / Name / Description: 00b — **[Default]** — Both Reference Indexes L0/L1 are forced to 0 during Concealment 01b — Only Reference Index L1 is forced to 0; Reference Index L0 is forced to -1 10b — Only Reference Index L0 is forced to 0; Reference Index L1 is forced to -1 11b — Reserved — Invalid
	26	**Reserved** Format: MBZ

Inline Data Description for MFD_AVC_BSD_Object

25 — MB Error Concealment B Temporal Motion Vectors Override Enable Flag

During MB Error Concealment on B slice with Temporal Direct Prediction, motion vectors are forced to 0 to improve image quality.

This bit can be set to preserve the original weight prediction.

Value	Name	Description
0	**[Default]**	Predicted Motion Vectors are used during MB Concealment
1		Motion Vectors are Overridden to 0 during MB Concealment

24 — MB Error Concealment B Temporal Weight Prediction Disable Flag

During MB Error Concealment on B slice with Temporal Direct Prediction, weight prediction is disabled to improve image quality.

This bit can be set to preserve the original weight prediction.

Value	Name	Description
0	**[Default]**	Weight Prediction is Disabled during MB Concealment
1		Weight Prediction will not be overridden during MB Concealment

23:22 — Reserved

Format:	MBZ

21:16 — Concealment Picture ID

This field identifies the picture in the reference list to be used for concealment. This field is only valid if **Concealment Method** is Inter P Copy.

Bit Filed	Value	Defenition
21	0	Frame Picture
21	1	Field picture
20:16	All	Frame Store Index[4:0]

15 — Reserved

Format:	MBZ

14 — BSD Premature Complete Error Handling

BSD Premature Complete Error occurs in situation where the Slice decode is completed but there are still data in the bitstream.

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W)
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically performs the error handling

13 — Reserved

Format:	MBZ

Inline Data Description for MFD_AVC_BSD_Object

12 — MPR Error (MV out of range) Handling

Software must follow the action for each Value as follow:

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W)
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically performs the error handling

11 — Reserved

Format:	MBZ

10 — Entropy Error Handling

Software must follow the action for each Value as follow:

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W).
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling.

9 — Reserved

Format:	MBZ

8 — MB Header Error Handling

Software must follow the action for each Value as follow:

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W).
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error concealment.

7:6 — MB Error Concealment B Spatial Prediction mode

These two bits control how the reference L0/L1 are overridden in B spatial slice.

Value	Name	Description
00b	**[Default]**	Both Reference Indexes L0/L1 are forced to 0 during Concealment
01b		Only Reference Index L1 is forced to 0; Reference Index L0 is forced to -1
10b		Only Reference Index L0 is forced to 0; Reference Index L1 is forced to -1
11b	Reserved	Invalid

5 — Reserved

Format:	MBZ

4 — MB Error Concealment B Spatial Motion Vectors Override Disable Flag

During MB Error Concealment on B slice with Spatial Direct Prediction, motion vectors are forced to 0 to improve image quality.

This bit can be set to use the predicted motion vectors instead.

This bit does not affect normal decoded MB.

Value	Name	Description
0	**[Default]**	Motion Vectors are Overridden to 0 during MB Concealment

Inline Data Description for MFD_AVC_BSD_Object

		1 | | Predicted Motion Vectors are used during MB Concealment
	3	**MB Error Concealment B Spatial Weight Prediction Disable Flag** During MB Error Concealment on B slice with Spatial Direct Prediction, weight prediction is disabled to improve image quality. This bit can be set to preserve the original weight prediction. This bit does not affect normal decoded MB. Value | Name | Description 0 | **[Default]** | Weight Prediction is Disabled during MB Concealment. 1 | | Weight Prediction will not be overridden during MB Concealment.
	2	**Reserved** Format: | MBZ
	1	**MB Error Concealment P Slice Motion Vectors Override Disable Flag** During MB Error Concealment on P slice, motion vectors are forced to 0 to improve image quality. This bit can be set to use the predicted motion vectors instead. This bit does not affect normal decoded MB. Value | Name | Description 0 | **[Default]** | Motion Vectors are Overridden to 0 during MB Concealment 1 | | Predicted Motion Vectors are used during MB Concealment
	0	**MB Error Concealment P Slice Weight Prediction Disable Flag** During MB Error Concealment on P slice, weight prediction is disabled to improve image quality. This bit can be set to preserve the original weight prediction. This bit does not affect normal decoded MB. Value | Name | Description 0 | **[Default]** | Weight Prediction is Disabled during MB Concealment. 1 | | Weight Prediction will not be overridden during MB Concealment.
1	31:16	**First MB Byte Offset of Slice Data or Slice Header** Programming Notes MFX supports only DXVA2 Long and Short Format.
	15:8	**Reserved** Format: | MBZ
	7	**Fix Prev Mb Skipped** Enables an alternative method for decoding mb_skipped, to cope with an encoder that codes a skipped MB as a direct MB with no coefficient.
	6:5	**Reserved** Format: | MBZ Programming Notes

Inline Data Description for MFD_AVC_BSD_Object

		Please note that the field MUST be set to '0' at this time.
	4	**Emulation Prevention Byte Present** Value / Name / Description 0 / / H/W needs to perform Emulation Byte Removal 1 / / H/W does not need to perform Emulation Byte Removal
	3	**LastSlice Flag** It is needed for both error concealment at the end of a picture (so, no more phantom slice. It is also needed to know to set the last MB in a picture correctly. Value / Name / Description 1 / / If the current Slice to be decoded is the very last slice of the current picture. 0 / / If the current Slice to be decoded is any slice other than the very last slice of the current picture
	2:0	**First Macroblock (MB)Bit Offset** Exists If: //AVC Long Format Only Format: U3 This field provides the bit offset of the first macroblock of the Slice in the first byte of the input compressed bitstream.
2	31	**I Slice Concealment Mode** This field controls how AVC decoder handle MB concealment in I Slice Value / Name 0 / Intra Concealment 1 / Inter Concealment **Programming Notes** If this field is set to "1" (Inter Concealment), driver must provide a valid reference picture (programmed using "Concealment Reference Picture" field) for concealment reference picture. In this mode, weight prediction is disabled and motion vectors are forced to 0 as well.
	30	**Reserved** Format: MBZ
	29:24	**Concealment Reference Picture + Field Bit** Format: U6 This field provides the concealment reference picture for hardware to conceal in case driver wants to specify one concealment picture. This field matches with the DPB order sent to hardware. This field applies to all I/P/B slices Bit Filed / Value / Defenition 29 / MBZ / is reserved for future expansion

Inline Data Description for MFD_AVC_BSD_Object

28:25	All	Reference PIcture Number
24	All	Field Bit(if the current picture is a field picture [Frame picture must be 0]

23 P Slice Concealment Mode

This field controls how AVC decoder handle MB concealment in P Slice

Value	Name
1	Intra Concealment
0	Inter Concealment

22:19 Reserved

Format:	MBZ

18:16 P Slice Inter Concealment Mode

This field controls how AVC decoder select reference picture for Concealment in P Slice.

Value	Name	Description
000b		Top of Reference List L0 (Use top entry of Reference List L0)
001b		Driver Specified Concealment Reference
010b		Predicted Reference (Use reference picture predicted using P-Skip Algorithm)
011b		Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC]
100b		First Long Term Picture in Reference List L0 (If no long term picture available, use Temporal Closest Picture)
101b-111b	Reserved	

15 B Slice Concealment Mode

This field controls how AVC decoder handle MB concealment in B Slice

Value	Name
1	Intra Concealment
0	Inter Concealment

14 Reserved

Format:	MBZ

13:12 B Slice Inter Direct Type Concealment Mode

AVC decoder can use Spatial or Temporal Direct for B Skip/Direct.

This field determine can override the mode on how AVC decoder handles MB concealment in B slice.

Value	Name	Description
00b		Use Default Direct Type (slice programmed direct type)
01b		Forced to Spatial Direct Only
10b		Forced to Temporal Direct Only

Inline Data Description for MFD_AVC_BSD_Object

11b		Spatial Direct without Temporal Componenet (MovingBlock information)

11 — Reserved

Format:	MBZ

10:8 — B Slice Spatial Inter Concealment Mode

This field controls how AVC decoder select reference picture for Spatial Inter Concealment in B Slice.

Value	Name	Description
000b		Top of Reference List L0/L1 (Use top entry of Reference List L0/L1).
001b		Driver Specified Concealment Reference
011b		Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC] [For L1: Closest POC larger than current POC]
100b		" First Long Term Picture in Reference List L0/L1 (If no long term picture available, use Temporal Closest Picture)
101b-111b	Reserved	

7 — Reserved

Format:	MBZ

6:4 — B Slice Temporal Inter Concealment Mode

This field controls how AVC decoder select reference picture for Temporal Inter Concealment in B Slice

Value	Name	Description
000b		Top of Reference List L0/L1 (Use top entry of Reference List L0/L1)
001b		Driver Specified Concealment Reference
010b		Predicted Reference (Use reference picture predicted using B-Skip Algorithm)
011b		" Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC] [For L1: Closest POC larger than current POC]
100b		First Long Term Picture in Reference List L0/L1 (If no long term picture available, use Temporal Closest Picture)
101b-111b	Reserved	

3:2 — Reserved

Format:	MBZ

1 — Intra 8x8/4x4 Prediction Error Concealment Control Bit

This field controls if AVC goes into MB concealment mode (next MB) when an error is detected on Intra8x8/4x4 Prediction Mode (these 2 modes have fixed coding so it may not affect the bitstream.

Inline Data Description for MFD_AVC_BSD_Object

Value	Name	Description
0		AVC decoder will NOT go into MB concealment when Intra8x8/4x4 Prediction mode is incorrect.
1		AVC decoder will go into MB concealment when Intra8x8/4x4 Prediction mode is incorrect.

0

Intra Prediction Error Control Bit (applied to Intra16x16/Intra8x8/Intra4x4 Luma and Chroma)

This field controls if AVC decoder will fix Intra Prediction Mode if the decoded value is incorrect according to MB position.

Value	Name	Description
0		AVC decoder will detect and fix Intra Prediction Mode Errors.
1		AVC decoder will retain the Intra Prediction value decoded from bitstream.

Inline Data Description - VP8 PAK OBJECT

Inline Data Description - VP8 PAK OBJECT

Source: VideoCS

Size (in bits): 384

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure corresponds to Dw3..6 of MFX_VP8_PAK_OBJECT Command.

DWord	Bit	Description
0	31:23	**Reserved** Format: MBZ
	22:20	**MV Format(Motion Vector Size)** Exists If: //IntraMbFlag = 0 This field specifies the size and format of the output motion vectors. Value / Name / Description: 000b – Intra MB – No Motion vectors 100b – Inter Predict MB (Unpacked Motion Vector Mode) – Sixteen Motion Vectors Per MacroBlock Others – Reserved **Programming Notes** This field MBZ, when the **IntraMbFlag = 1**.
	19:18	**SegmentID** Format: U2 Segment number 0-3
	17	**Enable Coeff Clamp** Value / Name / Description: 1 – Magnitude of coefficients of the current MB is clamped based on the clamping matrix after quantization 0 – No Clamping
	16:14	**Reserved** Format: MBZ
	13	**Intra MB Flag** This field specifies whether the current macroblock is an Intra (I) Macroblock. For Key pictures (IsKyeFrameFlag DW2, bit[5] of MFX_VP8_PIC_STATE), this field must be set to 1. Value / Name: 0h – INTER (Inter MacroBlock)

Inline Data Description - VP8 PAK OBJECT

		1h: INTRA (Intra MacroBlock)

Programming Notes
For I-picture MB (Intra MB Flag =1), this field must be set to 1.

12:11 **RefPicSelect**

This field specifies which reference pic (among Last Frame, Golden Frame and Alt Frame) is selected for the current macroblock when Intra MB Flag = 0 .

Value	Name
00b	Last Frame
01b	Golden Frame
10b	Alt Frame

10:8 **MB Type 3-Bits - Inter/Intra MB**

MB Type 3 Bits [10:8] specifies InterMB MV mode configurations: 16x16 or 2 16x8 or 4 8x8 or 16 4x4 when Intra MB Flag = 0 and bit [8] = IntraMB mode configurations: 4x4 or 16x16 when Intra MB Flag = 1

Value	Name	Description
000b	16x16	**Inter MB** Only DW 6 bits 3:0 are used to indicate MVMode, MVMode can't be split
001b	2 16x8 (mv_Top Bottom)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 16x8 partition, DW6 bits[3:0] are used for MVMode for second 16x8 partition.
010b	2 8 x16 (mv_left_right)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 8x16 partition, DW5 bits[11:8] are used for MVMode for second 8x16 partition.
011b	4 8x8 (mv_quarters)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 8x8 partition. DW5 bits[11:8] are used for MvMode for second 8x8 partition. DW6 bits[3:0] are used for MVMode for third 8x8 partition. DW6 bits[11:8] are used for MVMode for fourth 8x8 partition.
100b	16 4x4 (mv_16)	**Inter MB [10:8]** Split MV is inferred. There are 16 partitions. Each Sub-block uses 4 bits in DW6 and DW7.
0b	16x16	**Intra MB [8]** Only DW5, bits[3:0] are used for Y mode. For B_PRED, "16 4x4" should be used which implies B_PRED mode.
1b	16 4x4	**Intra MB [8]** All bits in DW5 and DW6 are used to represent B_PRED modes (Bmodes) in each sub-blocks.

Inline Data Description - VP8 PAK OBJECT

	7:6	**Reserved** Format: MBZ
	5:4	**MB UV Mode** Value / Name: 0 DC_PRED 1 V_PRED 2 H_PRED 3 TM_PRED
	3	**Reserved** Format: MBZ
	2	**Skip MB Flag** This field is equivalent to mb_skip_flag in VP8 spec. **Programming Notes** By setting this field to 1, it forces an Inter MacroBlock to be encoded as a skipped MacroBlock
	1:0	**Reserved** Format: MBZ
1	31:24	**Reserved** Format: MBZ
	23:16	**MbYCnt (Vertical Origin)** Format: U8 Unit of MacroBlock This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.
	15:8	**Reserved** Format: MBZ
	7:0	**MbXCnt (Horizontal Origin)** Format: U8 Unit of MacroBlock This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks.
2	31:28	**B Mode for SubBlock7 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	27:24	**B Mode for SubBlock6 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	23:20	**B Mode for SubBlock5 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	19:16	**B Mode for SubBlock4 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.

Inline Data Description - VP8 PAK OBJECT		
	15:12	**B Mode for SubBlock3 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	11:8	**B Mode for SubBlock2 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	7:4	**B Mode for SubBlock1 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	3:0	**B Mode for SubBlock0 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
3	31:28	**B Mode for SubBlock15 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	27:24	**B Mode for SubBlock14(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	23:20	**B Mode for SubBlock13(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	19:16	**B Mode for SubBlock12(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	15:12	**B Mode for SubBlock11(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	11:8	**B Mode for SubBlock10 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	7:4	**B Mode for SubBlock9 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	3:0	**B Mode for SubBlock8 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
4	31:16	**MV Y FWD 0** Format: U16 The value of the y component of this motion vector for FWD block 0.
	15:0	**MV X FWD 0** Format: U16 The value of the x component of this motion vector for FWD block 0.
5	31:16	**MV Y FWD 1** Format: U16 The value of the y component of this motion vector for FWD block 1.
	15:0	**MV X FWD 1** Format: U16 The value of the x component of this motion vector for FWD block 1.

Inline Data Description - VP8 PAK OBJECT		
6	31:16	**MV Y FWD 2** Format: U16 The value of the y component of this motion vector for FWD block 2.
	15:0	**MV X FWD 2** Format: U16 The value of the x component of this motion vector for FWD block 2.
7	31:16	**MV Y FWD 3** Format: U16 The value of the y component of this motion vector for FWD block 3.
	15:0	**MV X FWD 3** Format: U16 The value of the x component of this motion vector for FWD block 3.
8	31:16	**MV Y BWD 0** Format: U16 The value of the y component of this motion vector for BWD block 0.
	15:0	**MV X BWD 0** Format: U16 The value of the x component of this motion vector for BWD block 0.
9	31:16	**MV Y BWD 1** Format: U16 The value of the y component of this motion vector for BWD block 1.
	15:0	**MV X BWD 1** Format: U16 The value of the x component of this motion vector for BWD block 1.
10	31:16	**MV Y BWD 2** Format: U16 The value of the y component of this motion vector for BWD block 2.
	15:0	**MV X BWD 2** Format: U16 The value of the x component of this motion vector for BWD block 2.

Inline Data Description - VP8 PAK OBJECT		
11	31:16	**MV Y BWD 3** Format: U16 The value of the y component of this motion vector for BWD block 3.
	15:0	**MV X BWD 3** Format: U16 The value of the x component of this motion vector for BWD block 3.

INTERFACE_DESCRIPTOR_DATA

INTERFACE_DESCRIPTOR_DATA

Source: RenderCS

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:6	**Kernel Start Pointer** Format: InstructionBaseOffset[31:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel. This pointer is relative to the **Instruction Base Address.**
	5:0	**Reserved** Format: MBZ
1	31:16	**Reserved** Format: MBZ
	15:0	**Kernel Start Pointer High** Format: InstructionBaseOffset[47:32]Kernel This field specifies the high 16 bits of starting address of the Kernel Pointer.
2	31:20	**Reserved** Format: MBZ
	19	**Denorm Mode** This field specifies how Float denormalized numbers are handles in the dispatched thread. Value / Name / Description: 0h / Ftz / Float denorms will be flushed to zero when appearing as inputs; denorms will never come out of instructions. Double precision float and half precision float numbers are not flushed to zero. 1h / SetByKernel / Denorms will be handled in by kernel.
	18	**Single Program Flow** Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). Value / Name: 0h / Multiple 1h / Single
	17	**Thread Priority** Specifies the priority of the thread for dispatch. Value / Name:

INTERFACE_DESCRIPTOR_DATA

		0h: Normal Priority 1h: High Priority
	16	**Floating Point Mode** Specifies the floating point mode used by the dispatched thread. Value: 0h — Name: IEEE-754 Value: 1h — Name: Alternate
	15:14	**Reserved** Format: MBZ
	13	**Illegal Opcode Exception Enable** Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See *Exceptions and ISA Execution Environment*.
	12	**Reserved** Format: MBZ
	11	**Mask Stack Exception Enable** Format: Enable This bit gets loaded into EU CR0.1[11]. See *Exceptions and ISA Execution Environment*.
	10:8	**Reserved** Format: MBZ
	7	**Software Exception Enable** Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See *Exceptions and ISA Execution Environment*.
	6:0	**Reserved** Format: MBZ
3	31:5	**Sampler State Pointer** Format: DynamicStateOffset[31:5]SAMPLER_STATE Specifies the 32-byte aligned address offset of the sampler state table. This pointer is relative to the **Dynamic State Base Address.** *This field is ignored for child threads.*
	4:2	**Sampler Count** Format: U3 Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the

INTERFACE_DESCRIPTOR_DATA

associated sampler state entries.
This field is ignored for child threads.
If this field is not zero, sampler state is prefetched for the first instance of a root thread upon the startup of the media pipeline.

Value	Name
[0,4]	
0h	No samplers used
1h	Between 1 and 4 samplers used
2h	Between 5 and 8 samplers used
3h	Between 9 and 12 samplers used
4h	Between 13 and 16 samplers used

1:0 Reserved

Format:	MBZ

4 — 31:16 Reserved

Format:	MBZ

15:5 Binding Table Pointer

Format:	SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256

Description
Specifies the 32-byte aligned address of the binding table. This pointer is relative to the **Surface State Base Address**. *This field is ignored for child threads.*

4:0 Binding Table Entry Count

Format:	U5

Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state.
This field is ignored for child threads.
If this field is not zero, binding table and surface state are prefetched for the first instance of a root thread upon the startup of the media pipeline.

Value	Name
[0,31]	

Programming Notes
The maximum number of prefetched binding table entries is limited to 31. For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.

5 — 31:16 Constant/Indirect URB Entry Read Length

Format:	U16

Specifies the amount of URB data read and passed in the thread payload for the Constant or

INTERFACE_DESCRIPTOR_DATA

Indirect URB entry, in 8-DW register increments. A value 0 means that no Constant or Indirect URB Entry will be loaded. The Constant URB Entry Read Offset field will then be ignored. In GPGPU mode this describes how much data is delivered in a single dispatch. Multiple dispatches in a thread group will deliver constant data offset by this value. The total amount of constant data is (Constant URB Read Length * Number of Threads in GPGPU Thread Group + Cross-Thread Constant Data Read Length).

If **Cross-Thread Constant Data Read Length** for Indirect is greater than 0, then this field must also be greater than 0. The allowed combinations are:

Constant/Indirect URB Entry Read Length **Entry Read Length**	Cross-Thread Constant **Data Read Length**	Notes
=0	=0	No Payload
>0	=0	Per-thread payload only
>0	>0	Both kinds of payload
=0	>0	Only for CURBE payloads

Value	Name
[0,63]	

15:0 **Constant URB Entry Read Offset**

Format:	U16

Specifies the offset (in 8-DW units) at which Constant URB data is to be read from the URB before being included in the thread payload.

Value	Name	Description
[0,1983]		Indicating [0,1983] 256-bit register increments. ROB has 64KB of storage; 2048 entries. However, lowest 64 entries are reserved for VFE/TS to store interface descriptor data. Hence, (URB Entry Read Offset + Read Length) shall not exceed 1984.

6 31:24 **Reserved**

Format:	MBZ

23:22 **Rounding Mode**

Format:	U2

Value	Name	Description
00b	RTNE **[Default]**	Round to Nearest Even
01b	RU	Round toward +Infinity
10b	RD	Round toward -Infinity
11b	RTZ	Round toward Zero

INTERFACE_DESCRIPTOR_DATA

21 Barrier Enable

Format:	Enable

This field specifies whether the thread group requires a barrier. If not, it can be dispatched without allocating one.

20:16 Shared Local Memory Size

Format:	U5

This field indicates how much Shared Local Memory the thread group requires. The amount is specified in 4k blocks, but only powers of 2 are allowed: 0, 4k, 8k, 16k, 32k and 64k per half-slice.

Uses a different encoding to allow encodings for the new 1k and 2k SLM sizes.

Value	Name	Description
0	Encodes 0K	No SLM used
1	Encodes 1K	
2	Encodes 2K	
3	Encodes 4K	
4	Encodes 8K	
5	Encodes 16K	
6	Encodes 32K	
7	Encodes 64K	

15 Global Barrier Enable

Format:	Enable

Description
This field when set indicates that the thread group associated with this barrier is allowed to cross sub-slices with a performance penalty. When this field is clear, the thread group is dispatched to a single sub-slice or pool. Note that SLM should not be used with a Global Barrier since SLM is always forced to a single sub-slice or pool. The Barrier Enable bit must be set to enable the barrier, since this bit only specifies the type of barrier.
Restriction : Global barriers cannot be used. Must be zero.

14:13 Reserved

Format:	MBZ

12:10 Reserved

Format:	MBZ

INTERFACE_DESCRIPTOR_DATA

	9:0	**Number of Threads in GPGPU Thread Group** Format: U10 Specifies the number of threads that are in this thread group. The minimum value is 1, while the maximum value is the number of threads in a subslice for local barriers. See vol1b Configurations for the number of threads per subslice for different products. The maximum value for global barriers is limited by the number of threads in the system, or by 511, whichever is lower. This field should not be set to 0 even if the barrier is disabled, since an accurate value is needed for proper pre-emption.
7	31:8	**Reserved** Format: MBZ
	7:0	**Cross-Thread Constant Data Read Length** Format: U8 Specifies the amount of constant data in CURBE in 8-DW register increments which will be sent to every thread in the thread group in addition to the per thread ids specified by **Constant URB Entry Read Length**. **Value** [0,127] — **Name**

INTERRUPT

INTERRUPT

Source: BSpec
Access: RO, R/W, R/WC, R/W
Size (in bits): 128
Default Value: 0x00000000, 0xFFFFFFFF, 0x00000000, 0x00000000

See the Interrupt Definition Tables to find the source event for each interrupt bit.
There are multiple instances of this register format.

DWord	Bit	Description
0	31:0	**ISR** Access: RO These are the Interrupt Status Register Bits. This field contains the non-persistent values of the interrupt status bits. The IMR selects which of these interrupt conditions are reported in the persistent IIR **Value** / **Name** 0b – Condition Doesn't exist 1b – Condition Exists **Restriction** Restriction : Some inputs to this register are short pulses. Do not use this register to sample these conditions.
1	31:0	**IMR** Access: R/W These are the Interrupt Mask Register Bits. This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR. **Value** / **Name** FFFFFFFFh – All interrupts masked **[Default]** 0b – Not Masked 1b – Masked **Restriction** Restriction : For GT interrupts DO NOT use this register to mask interrupt events. Instead program this IMR to all 0s and use the individual GT command streamer MASK bits in the GT register space. This prevents unneeded messaging to DE.
2	31:0	**IIR** Access: R/WC These are the Interrupt Identity Register Bits.

<table>
<tr><th colspan="3">INTERRUPT</th></tr>
<tr><td></td><td></td><td>This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR.
The IER enables an interrupt to be generated when the corresponding bit in the IIR becomes set. A disabled interrupt will still appear in the IIR.
Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Condition Not Detected</td></tr>
<tr><td>1b</td><td>Condition Detected</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the first interrupt, the IIR bit will momentarily go low, then return high to indicate there is second interrupt pending.</td></tr>
</table></td></tr>
<tr><td>3</td><td>31:0</td><td>IER
<table>
<tr><td>Access:</td><td>R/W</td></tr>
</table>
These are the Interrupt Enable Register Bits.
The field enables an interrupt to be generated when the corresponding bit in the IIR becomes set. A disabled interrupt will still appear in the IIR.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The master interrupt enable must be set to 1b for any of these enabled interrupts to propagate to PCI device 2 interrupt processing.</td></tr>
</table></td></tr>
</table>

Invalidate After Read Message Descriptor Control Field

MDC_IAR - Invalidate After Read Message Descriptor Control Field

Source:	BSpec
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**Reserved** Format: MBZ Previously, this Enable field was intended to optimize scratch and spill/fill read messages, where the memory was only used by a single thread and did not need to be maintained after the thread completed. If enabled, it caused all lines in the L3 cache accessed by the message to be invalidated after the read occurred, regardless of whether the line contained modified data. It was intended as a performance hint indicating that the data would no longer be used to avoid writing back data to memory.

JPEG

JPEG		
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:5	**Reserved** Format: MBZ
	4	**Inconsistent VLD SE Error** This flag indicates an inconsistent SE coded in the bit-stream. Bit-stream does not match any entries in the hauffman table.
	3	**Extra Block Error** This flag indicates extra block coded within an ECS data boundary.
	2	**Missing block Error** This flag indicates one or more blocks are missing within an ECS data boundary.
	1	**Extra ECS Error** This flag indicates extra ECS' coded in the bit-stream SCAN payload data.
	0	**Missing ECS Error** This flag indicates one or more ECS' are missing from the bit-stream SCAN payload data.

LOD Message Address Payload Control

MACD_LOD - LOD Message Address Payload Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:4	**Reserved** Format: MBZ Ignored
	3:0	**LOD** Format: U4 Specifies the LOD for this slot. Value: [0,14] — Name: — Description: representing LOD

Lower Oword Block Data Payload

MDP_OW1L - Lower Oword Block Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword** Format: U128 Specifies the upper Oword data element
0.4-0.7	127:0	**Reserved** Format: Ignore Ignored

LRI Data Entry

LRI_DATA - LRI Data Entry

Source: RenderCS

Size (in bits): 64

Default Value: 0x00000000, 0x00000000

Each LRI command header is followed by LRI_DATA entries. Each of these entries is a pair of Dwords: the MMIO register address and the data to be written.

DWord	Bit	Description
0..1	63:55	**Reserved** Format: MBZ
	54:32	**MMIO** Format: U23 **Programming Notes** Bits [1:0] MBZ
	31:0	**Data** Format: U32

MEDIA_SURFACE_STATE

MEDIA_SURFACE_STATE

Source:	BSpec
Exists If:	//([MessageType] == 'Deinterlace') OR ([MessageType] == 'Sample_8x8')
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the SURFACE_STATE used by only deinterlace, sample_8x8, and VME messages.

DWord	Bit	Description
0	31:30	**Rotation** Value / Name: 00b — No Rotation or 0 Degree 01b — 90 Degree Rotation 10b — 180 Degree Rotation 11b — 270 Degree Rotation **Programming Notes** Rotation is only supported only with AVS function messages and not with HDC direct write and 16x8 AVS messages.
	29:27	**Reserved** Format: MBZ
	26:20	**X Offset** Exists If: //[Surface Format] is one of Planar Formats Format: PixelOffset[8:2] This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the surface. This field effectively loosens the alignment restrictions on the origin of tiled surfaces. Previously, tiled surface origin was (by definition) located at the base address, and thus needed to satisfy the 4KB base address alignment restriction. Now the origin can be specified at a finer (4-wide x 4-high pixel) resolution. Value / Name / Description: [0,508] — In multiples of 4 (low 2 bits missing) **Programming Notes** For linear surfaces and Packed Formats, this field must be zero. For **Surface Format** with 8 bits per element, this field must be a multiple of 16. For **Surface Format** with 16 bits per element, this field must be a multiple of 8.

MEDIA_SURFACE_STATE

	26:16	**Reserved** Exists If: //[Surface Format] is not one of Planar Formats Format: MBZ
	19:16	**Y Offset** Exists If: //[Surface Format] is one of Planar Formats Format: RowOffset[5:2] This field specifies the vertical offset in rows from the **Surface Base Address** to the start of the surface. (See additional description in the **X Offset** field) **Value** / **Name** / **Description**: [0,28] / / In multiples of 4 (low two bits missing) **Programming Notes**: For linear surfaces and Packed Formats, this field must be zero.
	15:12	**Reserved** Format: MBZ
	11:0	**Reserved** Format: MBZ
1	31:18	**Height** Format: U14-1 This field specifies the height of the surface in units of pixels. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. **Value** / **Name** / **Description** / **Exists If**: [0,16383] / / representing heights [1,16384] / [Surface Type] != FM_STRBUF_* **Programming Notes**: Height (field value + 1) must be a multiple of 2 for PLANAR_420 surfaces.If Vertical Line Stride is 1, this field indicates the height of the field, not the height of the frame.
	17:4	**Width** Format: U14-1 This field specifies the width of the surface in units of pixels. For PLANAR surface formats, this field indicates the width of the Y (luma) plane. **Value** / **Name** / **Description** / **Exists If**: [0,16383] / / representing widths [1,16384] / [Surface Type] != FM_STRBUF_* **Programming Notes**: • The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). • Width (field value + 1) must be a multiple of 2 for PLANAR_420, PLANAR_422, and all

MEDIA_SURFACE_STATE

YCRCB_* and Y16_UNORM surfaces, and must be a multiple of 4 for PLANAR_411 and Y8_UNORM_VA surfaces.

- For deinterlace messages, the Width (field value + 1) must be a multiple of 8.

- For Y8_UNORM_VA format width should be in multiple of 4, for Y16_UNORM_VA format width should be in multiple of 2, for Y1_UNORM format width should be in multiple of 32
- When Address Control = Mirror, the total width should be in multiple of 4bytes.

Width (field value + 1) must be a multiple of 2 for PLANAR_420_16

3:2 **Picture Structure**

Specifies the encoding of the current picture.

Value	Name
00b	Frame Picture
01b	Top Field Picture
10b	Bottom Field Picture
11b	Invalid, not allowed

1:0 **Cr(V)/Cb(U) Pixel Offset V Direction**

Default Value:	0
Format:	U0.2

Description
Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction

Programming Notes
This field is ignored for all formats except PLANAR_420_8
This offset has been increased from 2 bits to 3 bits to support U1.2 format, and the MSB bit is added as Pixel Offset V Direction MSB in DWord 2. Valid values for the combined field range from 0 to 4.

2 31:27 **Surface Format**

Description
Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.
Note: Y8_UNORM_VA, Y16_UNORM and Y16_SNORM are used for all functions of sample_8x8 except AVS where rest of the formats are not used. These two formats are packed as 32bits in L1 though the individual pixels are either 8bpp or 16bpp respectively.

Value	Name	Description

MEDIA_SURFACE_STATE

0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	
5	Y8_UNORM_VA	Sample_8x8 only except AVS
6	Y16_SNORM	Sample_8x8 only except AVS
7	Y16_UNORM_VA	Sample_8x8 only except AVS
8	R10G10B10A2_UNORM	Sample_8x8 only
9	R8G8B8A8_UNORM	Sample_8x8 AVS only
10	R8B8_UNORM (CrCb)	Sample_8x8 AVS only
11	R8_UNORM (Cr/Cb)	Sample_8x8 AVS only
12	Y8_UNORM	Sample_8x8 AVS only
13	A8Y8U8V8_UNORM	Sample_8x8 AVS only
14	B8G8R8A8_UNORM	Sample_8x8 AVS only
15	R16G16B16A16	Sample_8x8 AVS only
16	Y1_UNORM	Sample_8x8 only for boolean surfaces (1bit/pixel)
17	Y32_UNORM	For Integral Image (32bpp)
18	PLANAR_422_8	Sample_8x8 AVS only
Others	Reserved	

26 **Interleave Chroma**

Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.

25 **Cr(V)/Cb(U) Pixel Offset U Direction**

Default Value:	0
Format:	U0.1

Description

Specifies the distance to the U/V values with respect to the even numbered Y channels in the U direction

Programming Notes

This field must be zero for all formats except PLANAR_420_8, PLANAR_422_8, YCRCB_NORMAL, YCRCB_SWAPUVY, YCRCB_SWAPUV, YCRCB_SWAPY.

MEDIA_SURFACE_STATE

24 **Cr(V)/Cb(U) Pixel Offset V Direction MSB**

Default Value:	0
Format:	U1

Description
Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction

Programming Notes
This field must be zero for all formats except PLANAR_420_8.
This offset has been increased from 2 bits to 3 bits as U1.2 format and this bit is used in conjunction with the bits in the Cr(V)/Cb(U) Pixel Offset V Direction field in DWord 1, which contain the rest of the bits for offset V-direction. Valid values for the combined field range from 0 to 4.

23 **Memory Compression Mode**

Distinguishes Vertical from Horizontal compression.

Value	Name	Description
0	Horizontal Compression Mode **[Default]**	
1	Vertical Compression Mode	

22 **Memory Compression Enable**

Format:	Enable

This surface may contain compressed or compressible pixels. Memory compression will be attempted for writes to this surface. Reads from this surface will check for compressed data.

Programming Notes
The compression control must have 0 value for non-tileY modes.
Please refer to vol1a Memory Data Formats chapter -- section Media Memory Compression for more details, including format restrictions.

21 **Address Control**

Value	Name	Description
0	CLAMP	Clamp
1	MIRROR	Mirror

20:3 **Surface Pitch**

Format:	U18-1 pitch in Bytes

This field specifies the surface pitch in (#Bytes - 1).

Value	Name	Description
[0,262143]		For other linear surfaces: representing [1B, 256KB]
[511, 262143]		For X-tiled surface: representing [512B, 256KB] = [1 tile, 512 tiles]
[127, 262143]		For Y-tiled surfaces: representing [128B, 256KB] = [1 tile, 2048 tiles]

MEDIA_SURFACE_STATE

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile widthIf Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.The Surface Pitches of current picture and reference picture should be declared as the identical type in VDI mode with identical Height, Width and Format.
If Media Memory Compression is enabled, the following max pitch size restriction must be honored. For larger resolution, Media Memory compression Must be disabled. Tiling Mode Pixel Format Max Frame Width (bytes) Max Frame Width (pixels) Max Pitch (bytes) Legacy 4K 8bpp 16k 16k 16k + 127 16bpp 16k 8k 16k + 127 32bpp 16k 4k 16k + 127 64bpp 16k 2k 16k + 127 128bpp 16k 1k 16k + 127 TileYF 8bpp 8k 8k 8k + 63 16bpp 16k 8k 16k + 127 32bpp 16k 4k 16k + 127 64bpp 16k 2k 16k + 255 128bpp 16k 1k 16k + 255 TileYS 8bpp 16k 16k 16k + 255 16bpp 16k 8k 16k + 511 32bpp 16k 4k 16k + 511 64bpp 16k 2k 16k + 1023 128bpp 16k 1k 16k + 1023

2 **Half Pitch for Chroma**

Format:	Enable

This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field. This field is only used for PLANAR surface formats.

Programming Notes
Must be Zero as this field is not used.

1:0 **Tile Mode**

Format:	U2 Enumerated Type

This field specifies the type of memory tiling (Linear, WMajor, XMajor, or YMajor) employed to tile this surface. See Memory Interface Functions for details on memory tiling and restrictions.

Value	Name	Description
0h	TILEMODE_LINEAR	Linear mode (no tiling)
1h	Reserved	Reserved
2h	TILEMODE_XMAJOR	X major tiling
3h	TILEMODE_YMAJOR	Y major tiling

Programming Notes

- Refer to *Memory Data Formats* for restrictions on TileMode direction for the various buffer types. (Of particular interest is the fact that YMAJOR tiling is not supported for display/overlay buffers).
- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this field.
- Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled (X/Y/W) surfaces can only be mapped to Main Memory.

MEDIA_SURFACE_STATE

3	31:30	**Reserved** Format: MBZ
	29:16	**X Offset for U(Cb)** Format: U14 Pixel Offset **Description** For non planar surfaces this field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the surface. For Planar surfaces this field specifies the horizontal offset in pixels from the Y-plane origin to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. Resultant X-offset = 'X-offset of the surface (Y-plane)' + 'X offset for U(Cb)' For TileYS and TileYF this offset should be integral multiple of Tile width of Luma plane. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.
	15:14	**Reserved** Format: MBZ
	13:0	**Y Offset for U(Cb)** Format: U14 Row Offset **Description** For non planar surfaces this field specifies the vertical offset in pixels from the Surface Base Address to the start (origin) of the surface. For Planar surfaces this field specifies the vertical offset in rows from the Y-plane origin to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. Resultant X-offset = 'Y-offset of the surface (Y-plane)' + 'Y offset for U(Cb)' For TileYS and TileYF this offset should be integral multiple of Tile width of Luma plane. **Programming Notes** This field must be aligned by 4 bit[1:0] = 00 This field must be aligned by 4 bit[1:0] = 00 for all format besides PLANAR_420_*
4	31:30	**Reserved** Format: MBZ
	29:16	**X Offset for V(Cr)** Exists If: //([Surface Format] is one of planar) AND ([Interleave Chroma] == '0') Format: U14 Pixel Offset

MEDIA_SURFACE_STATE

		Description For Planar surfaces this field specifies the horizontal offset in pixels from the Y-plane origin to the start (origin) of the V(Cb) plane. Resultant X-offset = 'X-offset of the surface (Y-plane)' + 'X offset for V(Cb)' For TileYS and TileYF this offset should be integral multiple of Tile width of Luma plane. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for V(Cr)** Exists If: //([Surface Format] is one of planar) AND ([Interleave Chroma] == '0') Format: U15 Row Offset **Description** For Planar surfaces this field specifies the vertical offset in rows from the Y-plane origin to the start (origin) of the V(Cb) plane. Resultant Y-offset = 'Y-offset of the surface (Y-plane)' + 'Y offset for V(Cb)' For TileYS and TileYF this offset should be integral multiple of Tile width of Luma plane. **Programming Notes** This field must indicate a multiple of 4 (bit 0 & 1 = 00).
5	31	**Vertical Line Stride** Format: U1 in lines to skip between logically adjacent lines For Surfaces accessed via the sample_8x8 message:Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.For Other Surfaces:Vertical Line Stride must be zero.
	30	**Vertical Line Stride Offset** Format: U1 in lines of initial offset (when Vertical Line Stride == 1) For Surfaces accessed via the sample_8x8 message: Specifies the offset of the initial line from the beginning of the buffer. For Other Surfaces: Vertical Line Stride Offset must be zero. **Programming Notes** This field must be set to 0 if Vertical Line Stride is 0.
	29:24	**Reserved** Format: MBZ

MEDIA_SURFACE_STATE

	23:20	**Reserved** Format: MBZ
	19:18	**Tiled Resource Mode** Format: U2 **For Sampling Engine, Render Target, and Typed/Untyped Surfaces:** This field specifies the tiled resource mode. **For other surfaces:** This field is ignored.

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

Programming Notes
If **Tile Mode** is not set to TILEMODE_YMAJOR, this field must be set to TRMODE_NONE.
If this field is not set to TRMODE_NONE, the **Surface Format** must be one with 8, 16, 32, 64, or 128 bits per element, or one of the compressed texture modes (BC*, ETC*, EAC*, ASTC*). Additionally, YCRCB* formats are supported and treated as 16 bits per element, and the PLANAR_420_8 and PLANAR_422_8 formats are supported and treated as 8 bits per element on the Y plane and 16 bits per element on the UV plane (if **Interleave Chroma** is enabled) or 8 bits per element on the U and V planes (if **Interleave Chroma** is disabled.

	17:7	**Reserved** Format: MBZ
	6:0	**Surface Memory Object Control State** Default Value: 0h DefaultVaueDesc Format: **MEMORY_OBJECT_CONTROL_STATE** This 7-bit field is used in various state commands and indirect state objects to define cacheability and other attributes related to memory objects.
6	31:0	**Surface Base Address** Format: GraphicsAddress[31:0] Specifies the low 32 bits of the byte-aligned base address of the surface.

Programming Notes
For SURFTYPE_BUFFER render targets, this field specifies the base address of first element of the surface. The surface is interpreted as a simple array of that single element type. The address must be naturally-aligned to the element size (e.g., a buffer containing R32G32B32A32_FLOAT elements must be 16-byte aligned).For SURFTYPE_BUFFER non-rendertarget surfaces, this field specifies the base address of the first element of the surface, computed in software by adding the surface base address to the byte offset of the element in the buffer.Mipmapped, cube and

MEDIA_SURFACE_STATE

		3D sampling engine surfaces are stored in a 'monolithic' (fixed) format, and only require a single address for the base texture.Linear render target surface base addresses must be element-size aligned, for non-YUV surface formats, or a multiple of 2 element-sizes for YUV surface formats. Other linear surfaces have no alignment requirements (byte alignment is sufficient.)Linear depth buffer surface base addresses must be 64-byte aligned. Note that while render targets (color) can be SURFTYPE_BUFFER, depth buffers cannot.Tiled surface base addresses must be 4KB-aligned. Note that only the offsets from Surface Base Address are tiled, Surface Base Address itself is not transformed using the tiling algorithm.For tiled surfaces, the actual start of the surface can be offset from the Surface Base Address by the X Offset and Y Offset fields.Certain message types used to access surfaces have more stringent alignment requirements. Please refer to the specific message documentation for additional restrictions.
7	31:16	**Reserved** Format: MBZ
	15:0	**Surface Base Address High** Format: GraphicsAddress[47:32] Specifies the high 16 bits of the byte-aligned base address of the surface. Refer to Surface Base Address [31:0] for programming notes applying to this field.

MEMORY_OBJECT_CONTROL_STATE

MEMORY_OBJECT_CONTROL_STATE		
Source:	BSpec	
Size (in bits):	7	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	6:1	**Index to MOCS Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**

MemoryAddressAttributes

MemoryAddressAttributes

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. It defines the 32-bit memory address attributes for the third DWord of the HCP command buffer address.

DWord	Bit	Description
0	31:15	**Reserved** Format: MBZ
	14:13	**Base Address - Tiled Resource Mode** Format: U2 **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 00b / TRMODE_NONE / TileY resources 01b / TRMODE_TILEYF / 4KB tiled resources 10b / TRMODE_TILEYS / 64KB tiled resources 11b / Reserved /
	12	**Base Address - Row Store Scratch Buffer Cache Select** Format: U1 This field controls if the Row Store is going to store inside Media Cache (rowstore cache) or to LLC. Value / Name / Description: 0 / / Buffer going to LLC. 1 / / Buffer going to Internal Media Storage.
	11	**Reserved** Format: MBZ
	10	**Base Address - Memory Compression Mode** Format: U1 Distinguishes vertical from horizontal compression. Please refer to vol1a **Memory Data Formats chapter - section** media Memory Compression for more details. Value / Name: 0b / Horizontal Compression Mode **Programming Notes** Must be zero; vertical compression is not supported.

MemoryAddressAttributes		
	9	**Base Address - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface.
	8:7	**Base Address - Arbitration Priority Control** Format: HEVC_ARBITRATION_PRIORITY
	6:1	**Base Address - Index to Memory Object Control State (MOCS) Tables** Format: U6 The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**

Merged Media Block Message Header

MH_MBM - Merged Media Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Format: S31 X offset (in bytes) of the upper left corner of the block into the surface.
1	31:0	**Y Offset** Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Merged Media Block Message Control** Format: MHC_MBM_CONTROL Specifies the Merged message subtype and additional input parameters.
3	31:0	**Mask** Format: U32 The Mask is ignored by the Merged Media Block message: all Dwords are always returned on reads, and always enabled to be written on writes.
4	31:0	**FFTID** Format: MHC_FFTID Fixed Function Thread ID
5..7	95:0	**Reserved** Format: Ignore Ignored

Merged Media Block Message Header Control

MHC_MBM_CONTROL - Merged Media Block Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Format: Enumeration Specifies the Media Block Read message is Normal subtype. Value / Name / Description: 00h / Normal / The Block Height and Block Width fields are specified in this Dword. The Mask is ignofed by a media block read message. Others / Reserved / Reserved.
	29	**Reserved** Format: Ignore Ignored
	28:24	**Sub-Register Offset** Format: U5 Provides the sub-register offset in unit of bytes of a Merged Media Block Read message. This field is ignored (reserved) for a media block write message. Range = [0, 28]. Only a multiple of BasePitch, including 0, is valid. **Programming Notes** Sub-Register Offset and Register Pitch Control allow software to assembly multiple media block reads directly into a shared GRF register set. For example, if both are set to zero, the read data are written to GRF registers, aligning to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the Block Width. If Register Pitch Control is non-zero, multiple media block read messages sharing the same Register Pitch Control but with different Sub-Register Offset can fill in the same set of GRF registers with media block data line interleaved. **Restriction** Restriction : For the Sampler Cache Data, this field must be zero. Restriction : BasePitch is defined as the next the power-of-2 that is greater than or equal to the Block Width. Minimum BasePitch is 1 DWord. Sub-Register Offset must be aligned to BasePitch (therefore will be a multiple of DWords as well). When Register Pitch Control = 0, Sub-Register Offset must align to BasePitch*Block

MHC_MBM_CONTROL - Merged Media Block Message Header Control

		Height. ensuring the output fits in a single GRF register. In general (and specifically when Sub-Register Offset is greater than 0), when the resulting data will cross a GRF register boundary, the data must be placed symmetrically between GRF registers.
	23:22	**Reserved** Format: Ignore Ignored
	21:16	**Block Height** Format: U6 Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows **Restriction** Restriction : If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) * (# rows) <= 64 Dwords.
	15:10	**Reserved** Format: Ignore Ignored
	9:8	**Register Pitch Control** Format: U2 Controls the register pitch for a Merged Media Block Read message. This field is ignored (reserved) for a media block write message. Register Pitch Control is only allowed to be non-zero when Block Width is a multiple of DWords. Restriction : Register Pitch Control must be zero when SLICE_COMMON_ECO_CHICKEN1::DisableMediaMessageFix is set. Restriction : For the Sampler Cache Data, this field must be zero.

Value	Name	Description
0h	RPC_1 **[Default]**	1 Block
1h	RPC_2	2 Blocks
3h	RPC_4	4 Blocks

Restriction

Restriction :
BasePitch is defined as the next the power-of-2 that is greater than or equal to the Block Width. The effective register pitch (RPC*BasePitch)+SRO must be less than or equal to 32 bytes (to fit in a single GRF register).

MHC_MBM_CONTROL - Merged Media Block Message Header Control		
	7:6	**Reserved** Format: Ignore Ignored
	5:0	**Block Width** Format: U6 Width in bytes of the block being accessed. Range = [0,31] representing 1 to 32 Bytes.

Message Descriptor - Render Target Write

Message Descriptor - Render Target Write

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30	**Data Format** Format: U1 Value / Name / Description: 0 – Single Precision – 32b 1 – Half Precision – 16b **Programming Notes** This field is applicable for Render Target Write Messages ONLY.
	29:14	**Reserved** Format: MBZ
	13	**Per-Sample PS outputs enable** This bit must not be set when Render Target is not bound to pixel-shader OR when Render Target is not multisampled. This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. By setting this bit, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. When Render Target is multisampled and this bit is reset, Render Target outputs color, depth(optional) and stencil(optional) at pixel frequency. It should be noted that the latter case is applicable for only per-pixel PS invocation.
	12	**Last Render Target Select** This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. **Programming Notes** In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the antialias alpha, multisample coverage mask, and if the header is not present also includes the X/Y addresses and pixel enables. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set

Message Descriptor - Render Target Write

correctly for each message based on which slots are currently being processed.

Value	Name	Description
0	SLOTGRP_LO	choose bypassed data for slots 15:0
1	SLOTGRP_HI	choose bypassed data for slots 31:16

Programming Notes
For SIMD8 Image Write message thsi field MBZ.

10:8 **Message Type**

This field specifies the type of render target message.

For the SIMD8_DUALSRC_xx messages, the low bit indicates which slots to use for the pixel enables, X/Y addresses, and oMask.

Value	Name	Description
000b	SIMD16	SIMD16 single source message
001b	SIMD16_REPDATA	SIMD16 single source message with replicated data
010b	SIMD8_DUALSRC_LO	SIMD8 dual source message, use slots 7:0
011b	SIMD8_DUALSRC_HI	SIMD8 dual source message, use slots 15:8
100b	SIMD8_LO	SIMD8 single source message, use slots 7:0
111b	SIMD16_REPDATA_TM	It's only supported when accessing *Tiled Memory*. Using this Message Type to access linear *(Untiled)* memory is UNDEFINED.

Programming Notes
the above slots indicated are within the 16 slots selected by **Slot Group Select**. If SLOTGRP_HI is selected, the SIMD8 message types above reference slots 23:16 or 31:24 instead of 7:0 or 15:8, respectively.
SIMD16_REPDATA message must not be used in SIMD8 pixel-shaders.

7:0 **Reserved**

Format:	MBZ

Message Descriptor - Sampling Engine

Message Descriptor - Sampling Engine

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**EOT**
	30	**Return Format** Format: U1 Value / Name / Description: 0 / 32-bit / Return data is 32b 1 / 16-bit / Return data is 16b **Programming Notes** This field must be set to 32-bit for messages with SIMD Mode of SIMD4x2 or SIMD32/64. This field must be set to 32 for resinfo, LOD and sampleinfo messages. This message must be set to 32-bit for ld_mcs
	29	**Reserved** Format: MBZ
	28:25	**Message Length** Format: U4 This field specifies the number of 256-bit GRF registers starting from (src) to be sent out on the request message payload. Value / Name: [1,15] **Programming Notes** A value of 0 is considered erroneous.
	24:20	**Response Length** Format: U5 This field indicates the number of 256-bit registers expected in the message response. Value / Name: [0,16] **Programming Notes** A value 0 indicates that the request message does not expect any response. The largest

Message Descriptor - Sampling Engine

		response supported is 16 GRF registers.
	19	**Header Present** Format: Enable **Description** Specifies whether the message includes a header phase. If the header is not present (this field is zero), all of the fields normally contained in the header are assumed to be 0. If the header is not present, in some cases the **Write Channel Mask** fields are set according to the Response Length. For more details, please refer to the **Payload Parameter Definition**section, under *vol5c Shared Functions* .
	18:17	**SIMD Mode[1:0]** Format: U2 Specifies the SIMD mode of the message being sent. **Programming Notes** The SIMD Mode Extension field in the message header distinguishes between SIMD8D and SIMD4x2 modes.
	16:12	**Message Type** Format: U5 Specifies the type of message being sent. For more details, please refer to **Message Format** section for the definition of these 5 bits..
	11:8	**Sampler Index** Format: U4 Specifies the index into the sampler state table. Ignored for ld, resinfo, sampleinfo, and cache_flush type messages. **Value** / **Name**: [0,15] **Programming Notes** • For the deinterlace message, this field must be a multiple of 2 (even). • For the sample_8x8 message, this field must be a multiple of 4.
	7:0	**Binding Table Index** Format: U8 Specifies the index into the **binding table**. Ignored for cache_flush type messages. Values of 255 and 253 indicate stateless. 254 indicates SLM. 252 indicates bindless. **Value** / **Name**: [0,255]

MFD_MPEG2_BSD_OBJECT Inline Data Description

MFD_MPEG2_BSD_OBJECT Inline Data Description

Source: VideoCS
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DW0..1 corresponds to DW3..4 of the MFD_MPEG2_BSD_OBJECT.

DWord	Bit	Description
0	31:24	**Slice Horizontal Position** Format: U8 in Macroblocks This field indicates the horizontal position of the first macroblock in the slice.
	23:16	**Slice Vertical Position** Format: U8 in Macroblocks This field indicates the vertical position of the first macroblock in the slice.
	15:8	**Macroblock Count** Format: U8 in Macroblocks This field indicates the number of macroblocks in the slice, including skipped macroblocks.
	7	**Slice Concealment Override Bit** This bit forces hardware to handle the current slice in Conceal or Deocde Mode. If this bit is set to one, VIN will force the current slice to do concealment or to decode from bitstream regardless if the slice boundary has errors or not. Value / Name / Description: 1h — VIN will use driver-provided "Slice Concealment Type" regardless of valid slice boundary 0h — Driver must program "Slice Concealment Type" to '0'. VIN will set "Slice Concealment Type" depending if the slice boundary has error or not
	6	**Slice Concealment Type Bit** This bit can be forced by driver ("Slice Concealment Override Bit") or set by VINunit depending on slice boundary errors. Value / Name / Description: 1h — VMD will conceal all MBs of the slice regardless of bitstream. (If driver does not force the value of this bit, VIN will set this bit depending on slice boundary error. If the next slice position of the current slice is out-of-bound or the same or earlier than the current slice start position, VIN will set this bit for the next slice) 0h — VMD will decode MBs from the bitstream until the bitstream is run-out. Then VMD will conceal the remaining MBs. **Programming Notes** VIN can turn this bit from 0 to 1 internally if "Slice Concealment Disable Bit" is "0" and VIN detects slice boundary errors.

MFD_MPEG2_BSD_OBJECT Inline Data Description

	5	**Last Pic Slice** This bit is added to support error concealment at the end of a picture. **Value** / **Name** / **Description** 1h / / The current Slice is the last Slice of the entire picture 0h / / The current Slice is not the last Slice of current picture
	4	**Reserved**
	3	**Is Last MB** **Value** / **Name** / **Description** 1h / / The current MB is the last MB in the current Slice 0h / / The current MB is not the last MB in the current Slice
	2:0	**First Macroblock Bit Offset** Format: U3 This field provides the bit offset of the first macroblock in the first byte of the input bitstream.
1	31:29	**Reserved** Format: MBZ
	28:24	**Quantizer Scale Code** Format: U5 This field sets the quantizer scale code of the inverse quantizer. It remains in effect until changed by a decoded quantizer scale code in a macroblock. This field is decoded from the slice header by host software.
	23:17	**Reserved** Format: MBZ
	16:8	**Next Slice Vertical Position** Format: U9 in macroblocks This field indicates the vertical position (in macroblock units) of the first macroblock in the next slice. **Programming Notes** This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of the picture (field picture will be in height of field) (since y-direction is zero-based numbering).
	7:0	**Next Slice Horizontal Position** Format: U8 in macroblocks This field indicates the horizontal position (in macroblock units) of the first macroblock in the next slice. **Programming Notes** This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set 0.

MPEG2

MPEG2		
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:6	**Reserved** Format: MBZ
	5	**Missing EOB Error** This flag indicates missing EOB SEs coded in the bit-stream. Missing EOBs are concealed to match CBP of the error MB.
	4	**Inconsistent starting position Error - overlapping MBs** This flag indicates two slices overlapping one another by one or more MBs. Duplicate MBs decoded off the second slice shall be discarded.
	3	**Slice out-of-bound Error** This flag indicates a slice is running beyond the width of the picture. Out-of-bound MBs shall be discarded.
	2	**Premature frame end Error** This flag indicates missing slices/MBs coded in the bit-stream of a frame. One or more MBs are concealed to reach end of picture.
	1	**Inconsistent starting position Error - Missing MBs** This flag indicates one or more MBs are being concealed due to inconsistent MB starting and ending positions between slices.
	0	**MB Concealment Flag** . Each pulse from this flag indicates one MB is concealed by hardware.

MPEG4-2_Inline_DMEM

MPEG4-2_Inline_DMEM		
Source:	BSpec	
Size (in bits):	1600	
Default Value:	0x00000000, 0x00000000, 0x00000000,

MPEG4-2_Inline_DMEM		
6	31:1	**Reserved** Format: MBZ
	0	**hw_interlaced** MPEG4-2 SE: interlaced
7	31:1	**Reserved** Format: MBZ
	0	**hw_obmc_disable** MPEG4-2 SE: obmc_disable **Programming Notes** OBMC currently not supported; must be set to zero
8	31:1	**Reserved** Format: MBZ
	0	**hw_sprite_enable** MPEG4-2 SE: sprint_enable
9	31:6	**Reserved** Format: MBZ
	5:0	**hw_sprite_warping_points** MPEG4-2 SE: no_of_sprite_warping_points **Programming Notes** Only 0 and 1 supported
10	31:2	**Reserved** Format: MBZ
	1:0	**hw_sprite_warping_accuracy** MPEG4-2 SE: sprite_warping_accuracy **Programming Notes** Only 1/2, 1/4, 1/8 supported
11	31:4	**Reserved** Format: MBZ
	3:0	**hw_quant_precision** MPEG4-2 SE: quant_precision **Programming Notes** Must be set to 5h for ASP/SP profiles
12	31:1	**Reserved** Format: MBZ
	0	**hw_quant_type** MPEG4-2 SE: quant_type

MPEG4-2_Inline_DMEM		
13	31:1	**Reserved** Format: MBZ
	0	**hw_quarter_sample** MPEG4-2 SE: quarter_sample
14	31:1	**Reserved** Format: MBZ
	0	**hw_resync_marker_disable** MPEG4-2 SE: resync_marker_disable
15	31:1	**Reserved** Format: MBZ
	0	**hw_data_partitioned** MPEG4-2 SE: data_partitioned **Programming Notes** Data partitioning currently not supported; must be set to zero
16	31:1	**Reserved** Format: MBZ
	0	**hw_reversible_vlc** MPEG4-2 SE: reversible_vlc
17	31:7	**Reserved** Format: MBZ
	6:0	**hw_MacroBlockPerRow** Number of macroblocks per row, trunc(vop_width + 15 » 4)
18	31:7	**Reserved** Format: MBZ
	6:0	**hw_MacroBlockPerCol** Number of macroblocks per column, trunc(vop_height + 15 » 4)
19	31:15	**Reserved** Format: MBZ
	14:0	**hw_MacroBlockPerVOP** Number of macroblocks per VOP, MacroBlockPerRow * MacroBlockPerCol
20	31:4	**Reserved** Format: MBZ
	3:0	**hw_length_of_MB_number_code** Length of macroblock number code (1-14) in Table 6-27 column one of the MPEG4-2 standard specification
21	31:0	**hw_Tframe** Tframe calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification

MPEG4-2_Inline_DMEM		
22	31:0	**hw_TRD** TRD calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification
23	31:0	**hw_TRB** TRB calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification
24	31:2	**Reserved** Format: MBZ
	1:0	**hw_coding_type** MPEG4-2 SE: vop_coding_type
25	31:1	**Reserved** Format: MBZ
	0	**hw_rounding_type** MPEG4-2 SE: rounding_type
26	31:3	**Reserved** Format: MBZ
	2:0	**hw_intra_dc_vlc_thr** MPEG4-2 SE: intra_dc_vlc_thr
27	31:1	**Reserved** Format: MBZ
	0	**hw_top_field_first** MPEG4-2 SE: top_field_first
28	31:1	**Reserved** Format: MBZ
	0	**hw_alt_vertical_scan_flag** MPEG4-2 SE: alt_vertical_scan_flag
29	31:0	**hw_warping_mv_code_du[0]** MPEG4-2 SE: warping_mv_code(du[0])
30	31:0	**hw_warping_mv_code_du[1]** MPEG4-2 SE: warping_mv_code(du[1])
31	31:0	**hw_warping_mv_code_du[2]** MPEG4-2 SE: warping_mv_code(du[2])
32	31:0	**hw_warping_mv_code_du[3]** MPEG4-2 SE: warping_mv_code(du[3])
33	31:0	**hw_warping_mv_code_dv[0]** MPEG4-2 SE: warping_mv_code(dv[0])
34	31:0	**hw_warping_mv_code_dv[1]** MPEG4-2 SE: warping_mv_code(dv[1])

MPEG4-2_Inline_DMEM		
35	31:0	**hw_warping_mv_code_dv[2]** MPEG4-2 SE: warping_mv_code(dv[2])
36	31:0	**hw_warping_mv_code_dv[3]** MPEG4-2 SE: warping_mv_code(dv[3])
37	31:5	**Reserved** Format: MBZ
	4:0	**hw_quant** MPEG4-2 SE: vop_quant for non-short header mode only (see **hw_263_vop_quant** for short header mode)
38	31:3	**Reserved** Format: MBZ
	2:0	**hw_fcode_forward** MPEG4-2 SE: vop_fcode_forward
39	31:3	**Reserved** Format: MBZ
	2:0	**hw_fcode_backward** MPEG4-2 SE: vop_fcode_backward
40	31:9	**Reserved** Format: MBZ
	8:0	**hw_quant_scale** MPEG4-2 SE: quant_scale
41	31:8	**Reserved** Format: MBZ
	7:0	**hw_263_temporal_reference** MPEG4-2 SE: temporal_reference (short header format only)
42	31:5	**Reserved** Format: MBZ
	4:0	**hw_263_vop_quant** MPEG4-2 SE: vop_quant (short header format only)
43	31:5	**Reserved** Format: MBZ
	4:0	**hw_263_gob_number** MPEG4-2 SE: gob_number (short header format only)
44	31:0	**hw_263_num_gobs_in_vop** Derived from Table 6-29 in the MPEG4-2 standard specification (short header format only)
45	31:0	**hw_263_num_macroblocks_in_gob** Derived from Table 6-29 in the MPEG4-2 standard specification (short header format only)

MPEG4-2_Inline_DMEM		
46	31:1	**Reserved** Format: MBZ
	0	**hw_263_gob_header_empty** MPEG4-2 SE: gob_header_empty (short header format only)
47	31:1	**Reserved** Format: MBZ
	0	**hw_263_gob_frame_id** MPEG4-2 SE: gob_frame_id (short header format only)
48	31:9	**Reserved** Format: MBZ
	8:0	**hw_263_quant_scale** MPEG4-2 SE: quant_scale (short header format only)
49	31:0	**hw_263_num_rows_in_gob** Refer to Table 6-29 in the MPEG4-2 standard specification, vop_height/(16*num_gobs_in_vop) (short header format only)

MSAA Sample Number Message Address Control

MACD_MSAA_SN - MSAA Sample Number Message Address Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:4	**Reserved** Format: MBZ Ignored
	3:0	**Sample Number** Format: U4 Specifies the sample number for the slot. If the sample number is larger than the **Number of Multisamples** in the Surface State, then the access is out of bounds.

MsgDescpt31

MsgDescpt31

Source: Eulsa
Size (in bits): 29
Default Value: 0x00000000

DWord	Bit	Description
0	28:25	**Message Length** This field specifies the number of 256-bit MRF registers starting from <curr_dest> to be sent out on the request message payload. Valid value ranges from 1 to 15. A value of 0 is considered erroneous. Value: 1-15 — Name: Number of MRF Registers
	24:20	**Response Length** This field indicates the number of 256-bit registers expected in the message response. The valid value ranges from 0 to 16. A value 0 indicates that the request message does not expect any response. The largest response supported is 16 GRF registers. Value: 0-16 — Name: Number of Registers
	19	**Header Present** Format: Enable If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details.
	18:0	**Function Control** This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.

Normal Media Block Message Header

MH_MB - Normal Media Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** Must be DWord aligned (Bits 1:0 MBZ) for the write form of the message.
1	31:0	**Y Offset** Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Normal Media Block Message Control** Format: MHC_MB_CONTROL Specifies the Normal message subtype and additional input parameters.
3	31:0	**Mask** Format: U32 The Mask is ignored by the Normal Media Block message: all Dwords are always returned on reads, and always enabled to be written on writes.
4	31:0	**FFTID** Format: MHC_FFTID Fixed Function Thread ID
5..7	95:0	**Reserved** Format: Ignore Ignored

Normal Media Block Message Header Control

MHC_MB_CONTROL - Normal Media Block Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Format: Enumeration Specifies the interpretation of M0.3 (Pixel or Byte Mask). For the Sampler Cache Data Port, this field is ignored, behaving as if always set to NORMAL. **Value / Name / Description** 00h / Normal / The Block Height and Block Width fields are specified in this Dword. The Mask is ignored by a media block read message and behaves as if it is set to all ones for a media block write message. Others / Reserved / Reserved. **Programming Notes** The Media Block Read message is Normal subtype when both Sub-Register Offset and Register Pitch Control are zero. The Media Block Read message is Merged subtype when either Sub-Register Offset or Register Pitch Control are non-zero.
	29	**Reserved** Format: Ignore Ignored
	28:24	**Sub-Register Offset** Format: MBZ The sub-register offset must be 0 for Normal Media Block Read message subtype. This field is ignored (reserved) for a media block write message.
	23:22	**Reserved** Format: Ignore Ignored

MHC_MB_CONTROL - Normal Media Block Message Header Control

	21:16	**Block Height** Format: U6 Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows **Restriction** Restriction : If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) * (# rows) <= 64 Dwords.
	15:10	**Reserved** Format: Ignore Ignored
	9:8	**Register Pitch Control** Format: MBZ The register pitch must be 0 for a Normal Media Block Read message. This field is ignored (reserved) for a media block write message.
	7:6	**Reserved** Format: Ignore Ignored
	5:0	**Block Width** Format: U6 Width in bytes of the block being accessed. For normal Media Block Writes, Range = [0,63] representing 1 to 64 Bytes. For normal Media Block Reads and for masked and merged Media Block messages, Range = [0,31] representing 1 to 32 Bytes. **Programming Notes** Must be DWord aligned for the write form of the message.

oMask Message Data Payload Register

MDPR_OMASK - oMask Message Data Payload Register

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**oMask1** Format: U16 oMask for Pixels [15:0] of Slot 1. Not used for Slot Group HI.
	15:0	**oMask0** Format: U16 oMask for Pixels [15:0] of Slot 0. Not used for Slot Group HI.
1	31:16	**oMask3** Format: U16 oMask for Pixels [15:0] of Slot 3. Not used for Slot Group HI.
	15:0	**oMask2** Format: U16 oMask for Pixels [15:0] of Slot 2. Not used for Slot Group HI.
2	31:16	**oMask5** Format: U16 oMask for Pixels [15:0] of Slot 5. Not used for Slot Group HI.
	15:0	**oMask4** Format: U16 oMask for Pixels [15:0] of Slot 4. Not used for Slot Group HI.
3	31:16	**oMask7** Format: U16 oMask for Pixels [15:0] of Slot 7. Not used for Slot Group HI.
	15:0	**oMask6** Format: U16 oMask for Pixels [15:0] of Slot 6. Not used for Slot Group HI.

MDPR_OMASK - oMask Message Data Payload Register		
4	31:16	**oMask9** Format: U16 oMask for Pixels [15:0] of Slot 9. Used only if Slot Group HI or SIMD16.
	15:0	**oMask8** Format: U16 oMask for Pixels [15:0] of Slot 8. Used only if Slot Group HI or SIMD16.
5	31:16	**oMask11** Format: U16 oMask for Pixels [15:0] of Slot 11. Used only if Slot Group HI or SIMD16.
	15:0	**oMask10** Format: U16 oMask for Pixels [15:0] of Slot 10. Used only if Slot Group HI or SIMD16.
6	31:16	**oMask13** Format: U16 oMask for Pixels [15:0] of Slot 13. Used only if Slot Group HI or SIMD16.
	15:0	**oMask12** Format: U16 oMask for Pixels [15:0] of Slot 12. Used only if Slot Group HI or SIMD16.
7	31:16	**oMask15** Format: U16 oMask for Pixels [15:0] of Slot 15. Used only if Slot Group HI or SIMD16.
	15:0	**oMask14** Format: U16 oMask for Pixels [15:0] of Slot 14. Used only if Slot Group HI or SIMD16.

OM Replicated SIMD16 Render Target Data Payload

MDP_RTW_M16REP - OM Replicated SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK Slots [15:0] oMask
1.0-1.7	255:0	**RGBA** Format: MDPR_RGBA RGBA for all slots [15:0]

OM S0A SIMD8 Render Target Data Payload

MDP_RTW_MA8 - OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

OM S0A SIMD16 Render Target Data Payload

MDP_RTW_MA16 - OM S0A SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 2816

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_MA16 - OM S0A SIMD16 Render Target Data Payload		
5.0-5.7	255:0	**Green[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Green
6.0-6.7	255:0	**Green[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Green
7.0-7.7	255:0	**Blue[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Blue
8.0-8.7	255:0	**Blue[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
9.0-9.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
10.0-10.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha

OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload		
6.0-6.7	255:0	**Src1 Green** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha

OM SIMD8 Render Target Data Payload

MDP_RTW_M8 - OM SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Format: MDPR_OMASK Slots [7:0] oMask. Upper half ignored.
1.0-1.7	255:0	**Red** Format: MDP_DW_SIMD8 Slots [7:0] Red
2.0-2.7	255:0	**Green** Format: MDP_DW_SIMD8 Slots [7:0] Green
3.0-3.7	255:0	**Blue** Format: MDP_DW_SIMD8 Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha

OM SIMD16 Render Target Data Payload

MDP_RTW_M16 - OM SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_M16 - OM SIMD16 Render Target Data Payload		
6.0-6.7	255:0	**Blue[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
7.0-7.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
8.0-8.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha

OS OM S0A SIMD8 Render Target Data Payload

MDP_RTW_SMA8 - OS OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1792
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SMA8 - OS OM S0A SIMD8 Render Target Data Payload		
6.0-6.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

OS OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_SM8DS - OS OM SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	2560
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SM8DS - OS OM SIMD8 Dual Source Render Target Data Payload

5.0-5.7	255:0	**Src1 Red** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Red
6.0-6.7	255:0	**Src1 Green** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Alpha
9.0-9.7	255:0	**Stencil** Format: **MDPR_STENCIL** Slots [7:0] or [15:8] of Stencil

OS OM SIMD8 Render Target Data Payload

MDP_RTW_SM8 - OS OM SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

OS S0A SIMD8 Render Target Data Payload

MDP_RTW_SA8 - OS S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000, 0x00000000,

OS SIMD8 Dual Source Render Target Data Payload

MDP_RTW_S8DS - OS SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_S8DS - OS SIMD8 Dual Source Render Target Data Payload

5.0-5.7	255:0	**Src1 Green** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Green
6.0-6.7	255:0	**Src1 Blue** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Format: **MDP_DW_SIMD8** Slots[7:0] or [15:8] of Src1 Alpha
8.0-8.7	255:0	**Stencil** Format: **MDPR_STENCIL** Slots [7:0] or [15:8] of Stencil

OS SIMD8 Render Target Data Payload

MDP_RTW_S8 - OS SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDP_DW_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDP_DW_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDP_DW_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
4.0-4.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

OS SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTW_SZMA8 - OS SZ OM S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SZMA8 - OS SZ OM S0A SIMD8 Render Target Data Payload

5.0-5.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
6.0-6.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth
7.0-7.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

OS SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_SZM8DS - OS SZ OM SIMD8 Dual Source Render Target Data Payload

Source: BSpec

Size (in bits): 2816

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SZM8DS - OS SZ OM SIMD8 Dual Source Render Target Data Payload

5.0-5.7	255:0	**Src1 Red** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Red
6.0-6.7	255:0	**Src1 Green** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha
9.0-9.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] or [15:8] of Source Depth
10.0-10.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] or [15:8] of Stencil

OS SZ OM SIMD8 Render Target Data Payload

MDP_RTW_SZM8 - OS SZ OM SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SZM8 - OS SZ OM SIMD8 Render Target Data Payload		
6.0-6.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

OS SZ S0A SIMD8 Render Target Data Payload

MDP_RTW_SZA8 - OS SZ S0A SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SZA8 - OS SZ S0A SIMD8 Render Target Data Payload		
6.0-6.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] Stencil

OS SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTW_SZ8DS - OS SZ SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	2560
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_SZ8DS - OS SZ SIMD8 Dual Source Render Target Data Payload

5.0-5.7	255:0	**Src1 Green** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Green
6.0-6.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha
8.0-8.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] or [15:8] of Source Depth
9.0-9.7	255:0	**Stencil** Format: MDPR_STENCIL Slots [7:0] or [15:8] of Stencil

OS SZ SIMD8 Render Target Data Payload

MDP_RTW_SZ8 - OS SZ SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1536

Default Value: 0x00000000, 0x00000000, 0x00000000,

Oword 1 Dual Block Data Payload

MDP_OWD1 - Oword 1 Dual Block Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword Slot0** Format: U128 Specifies the Slot 0 data
0.4-0.7	127:0	**Oword Slot1** Format: U128 Specifies the Slot 1 data

Oword 2 Block Data Payload

MDP_OW2 - Oword 2 Block Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword0** Format: U128 Specifies the Oword data for block element 0
0.4-0.7	127:0	**Oword1** Format: U128 Specifies the Oword data for block element 1

Oword 4 Block Data Payload

MDP_OW4 - Oword 4 Block Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[1:0]** Format: MDCR_OW Specifies the Oword data for block elements [1:0]
1.0-1.7	255:0	**Data[3:2]** Format: MDCR_OW Specifies the Oword data for block elements [3:2]

Oword 4 Dual Block Data Payload

MDP_OWD4 - Oword 4 Dual Block Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword0 Slot0** Format: U128 Specifies the Slot 0 data for block element 0
0.4-0.7	127:0	**Oword0 Slot1** Format: U128 Specifies the Slot 1 data for block element 0
1.0-1.3	127:0	**Oword1 Slot0** Format: U128 Specifies the Slot 0 data for block element 1
1.4-1.7	127:0	**Oword1 Slot1** Format: U128 Specifies the Slot 1 data for block element 1
2.0-2.3	127:0	**Oword2 Slot0** Format: U128 Specifies the Slot 0 data for block element 2
2.4-2.7	127:0	**Oword2 Slot1** Format: U128 Specifies the Slot 1 data for block element 2
3.0-3.3	127:0	**Oword3 Slot0** Format: U128 Specifies the Slot 0 data for block element 3

MDP_OWD4 - Oword 4 Dual Block Data Payload		
3.4-3.7	127:0	**Oword3 Slot1** Format: U128 Specifies the Slot 1 data for block element 3

Oword 8 Block Data Payload

MDP_OW8 - Oword 8 Block Data Payload		
Source:	BSpec	
Size (in bits):	1024	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Data[1:0]** Format: MDCR_OW Specifies the Oword data for block elements [1:0]
1.0-1.7	255:0	**Data[3:2]** Format: MDCR_OW Specifies the Oword data for block elements [3:2]
2.0-2.7	255:0	**Data[5:4]** Format: MDCR_OW Specifies the Oword data for block elements [5:4]
3.0-3.7	255:0	**Data[7:6]** Format: MDCR_OW Specifies the Oword data for block elements [7:6]

Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload

MDP_A64_AOP4X2_OW2 - Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Src0 Slot0** Format: U128 Specifies the Slot 0 Source 0 data
0.4-0.7	127:0	**Src0 Slot1** Format: U128 Specifies the Slot 1 Source 0 data
1.0-1.3	127:0	**Src1 Slot0** Format: U128 Specifies the Slot 0 Source 1 data
1.4-1.7	127:0	**Src1 Slot1** Format: U128 Specifies the Slot 1 Source 1 data

Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload

MDP_A64_AOP4X2_OW1 - Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword0** Format: U128 Specifies the Slot 0 Return data
0.4-0.7	127:0	**Oword1** Format: U128 Specifies the Slot1 Return data

Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload

MDP_A64_AOP8_OW2 - Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_A64_AOP8_OW2 - Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload		
5.0-5.7	255:0	**Slot[3:2] Src1** Format: MDCR_OW Specifies the Slot [3:2] Source 1 data
6.0-6.7	255:0	**Slot[5:4] Src1** Format: MDCR_OW Specifies the Slot [5:4] Source 1 data
7.0-7.7	255:0	**Slot[7:6] Src1** Format: MDCR_OW Specifies the Slot [7:6] Source 1 data

Oword Data Blocks Message Descriptor Control Field

MDC_DB_OW - Oword Data Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2:0	**Data Blocks** Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description
00h	OW1L	1 Oword, read into or written from the low 128 bits of the destination register
01h	OW1U	1 Oword, read into or written from the high 128 bits of the destination register
02h	OW2	2 Owords
03h	OW4	4 Owords
04h	OW8	8 Owords
Others	Reserved	Ignored

Oword Data Payload Register

MDCR_OW - Oword Data Payload Register		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword0** Format: U128 Specifies the slot 0 data in this payload register
0.4-0.7	127:0	**Oword1** Format: U128 Specifies the slot 1 data in this payload register

Oword Dual Data Blocks Message Descriptor Control Field

MDC_DB_OWD - Oword Dual Data Blocks Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**OW Dual Data Blocks** Format: Enumeration Specifies the number of Oword Blocks to be read or written

Value	Name	Description
00h	OWD1	1 Hword register, 2 Owords
02h	OWD4	4 Hword registers, 8 Owords
Others	Reserved	Ignored

PALETTE_ENTRY

PALETTE_ENTRY		
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:24	**Alpha** Format: U8 Alpha channel value for this entry in the texture color palette.
	23:16	**Red** Format: U8 Red channel value for this entry in the texture color palette.
	15:8	**Green** Format: U8 Green channel value for this entry in the texture color palette.
	7:0	**Blue** Format: U8 Blue channel value for this entry in the texture color palette.

Performance Counter Report Format 101b

Performance Counter Report Format 101b		
Source:	BSpec	
Size (in bits):	2048	
Default Value:	0x00000000, 0x00000000, 0x00000000,

Performance Counter Report Format 101b		
21	31:0	**A-Cntr 17 (low dword)**
22	31:0	**A-Cntr 18 (low dword)**
23	31:0	**A-Cntr 19 (low dword)**
24	31:0	**A-Cntr 20 (low dword)**
25	31:0	**A-Cntr 21 (low dword)**
26	31:0	**A-Cntr 22 (low dword)**
27	31:0	**A-Cntr 23 (low dword)**
28	31:0	**A-Cntr 24 (low dword)**
29	31:0	**A-Cntr 25 (low dword)**
30	31:0	**A-Cntr 26 (low dword)**
31	31:0	**A-Cntr 27 (low dword)**
32	31:0	**A-Cntr 28 (low dword)**
33	31:0	**A-Cntr 29 (low dword)**
34	31:0	**A-Cntr 30 (low dword)**
35	31:0	**A-Cntr 31 (low dword)**
36	31:0	**A-Cntr 32 (low dword)**
37	31:0	**A-Cntr 33 (low dword)**
38	31:0	**A-Cntr 34 (low dword)**
39	31:0	**A-Cntr 35 (low dword)**
40	31:24	**High byte of A3**
	23:16	**High byte of A2**
	15:8	**High byte of A1**
	7:0	**High byte of A0**
41	31:24	**High byte of A7**
	23:16	**High byte of A6**
	15:8	**High byte of A5**
	7:0	**High byte of A4**
42	31:24	**High byte of A11**
	23:16	**High byte of A10**
	15:8	**High byte of A9**
	7:0	**High byte of A8**
43	31:24	**High byte of A15**
	23:16	**High byte of A14**
	15:8	**High byte of A13**
	7:0	**High byte of A12**
44	31:24	**High byte of A19**

Performance Counter Report Format 101b		
	23:16	**High byte of A18**
	15:8	**High byte of A17**
	7:0	**High byte of A16**
45	31:24	**High byte of A23**
	23:16	**High byte of A22**
	15:8	**High byte of A21**
	7:0	**High byte of A20**
46	31:24	**High byte of A27**
	23:16	**High byte of A26**
	15:8	**High byte of A25**
	7:0	**High byte of A24**
47	31:24	**High byte of A31**
	23:16	**High byte of A30**
	15:8	**High byte of A29**
	7:0	**High byte of A28**
48	31:0	**B-Cntr 0**
49	31:0	**B-Cntr 1**
50	31:0	**B-Cntr 2**
51	31:0	**B-Cntr 3**
52	31:0	**B-Cntr 4**
53	31:0	**B-Cntr 5**
54	31:0	**B-Cntr 6**
55	31:0	**B-Cntr 7**
56	31:0	**C-Cntr 0**
57	31:0	**C-Cntr 1**
58	31:0	**C-Cntr 2**
59	31:0	**C-Cntr 3**
60	31:0	**C-Cntr 4**
61	31:0	**C-Cntr 5**
62	31:0	**C-Cntr 6**
63	31:0	**C-Cntr 7**

Per Thread Scratch Space Message Header Control

MHC_PTSS - Per Thread Scratch Space Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:4	**Reserved** Format: Ignore Ignored
	3:0	**Per Thread Scratch Space** Format: U4 Specifies the amount of scratch space allowed to be used by this thread for messages in which the Binding Table Index is Stateless model, otherwise this field is ignored.The data port will use this to bounds check scratch space messages. Value range = [0,11] represents [1KB, 2MB] in powers of two. **Programming Notes** Writes out of bounds will be ignored. Reads out of bounds will return 0.

PIXEL_HASH_TABLE_1BIT_32ENTRY

PIXEL_HASH_TABLE_1BIT_32ENTRY

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

Description
2-way pixel hashing table. Table is 32-entries:8X,4Y in [Y][X] format. Each entry is a single bit that indicates which sub-slice hardware block the indicated xy pixel block is mapped.
pixelhash_id maps to subslice. A value of 0 indicates the first enabled subslice. A value of 1 indicates the second enabled subslice.

DWord	Bit	Description
0	31:24	**Pixel Hashing Table Entries y[3]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=3 and x=7..0
	23:16	**Pixel Hashing Table Entries y[2]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=2 and x=7..0
	15:8	**Pixel Hashing Table Entries y[1]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=1 and x=7..0
	7:0	**Pixel Hashing Table Entries y[0]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=0 and x=7..0

PIXEL_HASH_TABLE_1BIT_64ENTRY

PIXEL_HASH_TABLE_1BIT_64ENTRY	
Source:	BSpec
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

Description
2-way pixel hashing table. Table is 64-entries:8X,8Y in [Y][X] format. Each entry is a single bit that indicates which sub-slice hardware block the indicated xy pixel block is mapped.
pixelhash_id maps to subslice. A value of 0 indicates the first enabled subslice. A value of 1 indicates the second enabled subslice.

DWord	Bit	Description
0	31:24	**Pixel Hashing Table Entries y[3]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=3 and x=7..0
	23:16	**Pixel Hashing Table Entries y[2]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=2 and x=7..0
	15:8	**Pixel Hashing Table Entries y[1]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=1 and x=7..0
	7:0	**Pixel Hashing Table Entries y[0]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=0 and x=7..0
1	31:24	**Pixel Hashing Table Entries y[7]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=7 and x=7..0
	23:16	**Pixel Hashing Table Entries y[6]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=6 and x=7..0
	15:8	**Pixel Hashing Table Entries y[5]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=5 and x=7..0

PIXEL_HASH_TABLE_1BIT_64ENTRY		
	7:0	**Pixel Hashing Table Entries y[4]x[7:0]** Format: U8 Indicates the pixelhash_id for the pixel block that has y=4 and x=7..0

PIXEL_HASH_TABLE_1BIT_128ENTRY

PIXEL_HASH_TABLE_1BIT_128ENTRY		
Source:	BSpec	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
Description		
2-way pixel hashing table. Table is 128-entries:16X,8Y in [Y][X] format. Each entry is a single bit that indicates which sub-slice hardware block the indicated xy pixel block is mapped.		
pixelhash_id maps to subslice. A value of 0 indicates the first enabled subslice. A value of 1 indicates the second enabled subslice.		

DWord	Bit	Description
0	31:16	**Pixel Hashing Table Entries y[1]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=1 and x=15..0
	15:0	**Pixel Hashing Table Entries y[0]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=0 and x=15..0
1	31:16	**Pixel Hashing Table Entries y[3]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=3 and x=15..0
	15:0	**Pixel Hashing Table Entries y[2]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=2 and x=15..0
2	31:16	**Pixel Hashing Table Entries y[5]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=5 and x=15..0
	15:0	**Pixel Hashing Table Entries y[4]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=4 and x=15..0
3	31:16	**Pixel Hashing Table Entries y[7]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=7 and x=15..0

PIXEL_HASH_TABLE_1BIT_128ENTRY		
	15:0	**Pixel Hashing Table Entries y[6]x[15:0]** Format: U16 Indicates the pixelhash_id for the pixel block that has y=6 and x=15..0

PIXEL_HASH_TABLE_2BIT_64ENTRY

PIXEL_HASH_TABLE_2BIT_64ENTRY	
Source:	BSpec
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

Description
3-wayor 4-way pixel hashing table. Table is 64-entries:8X,8Y in [Y][X] format. Each entry is two bits that indicates which sub-slice hardware block the indicated xy pixel block is mapped.
pixelhash_id maps to subslice. A value of 0 indicates the first enabled subslice. A value of 1 indicates the second enabled subslice.

DWord	Bit	Description
0	31:30	**Pixel Hashing Table Entry y[1]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=1
	29:28	**Pixel Hashing Table Entry y[1]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=1
	27:26	**Pixel Hashing Table Entry y[1]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=1
	25:24	**Pixel Hashing Table Entry y[1]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=1
	23:22	**Pixel Hashing Table Entry y[1]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=1
	21:20	**Pixel Hashing Table Entry y[1]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=1
	19:18	**Pixel Hashing Table Entry y[1]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=1

PIXEL_HASH_TABLE_2BIT_64ENTRY

	17:16	**Pixel Hashing Table Entry y[1]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=1
	15:14	**Pixel Hashing Table Entry y[0]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=0
	13:12	**Pixel Hashing Table Entry y[0]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=0
	11:10	**Pixel Hashing Table Entry y[0]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=0
	9:8	**Pixel Hashing Table Entry y[0]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=0
	7:6	**Pixel Hashing Table Entry y[0]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=0
	5:4	**Pixel Hashing Table Entry y[0]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=0
	3:2	**Pixel Hashing Table Entry y[0]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=0
	1:0	**Pixel Hashing Table Entry y[0]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=0
1	31:30	**Pixel Hashing Table Entry y[3]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=3

PIXEL_HASH_TABLE_2BIT_64ENTRY

	29:28	**Pixel Hashing Table Entry y[3]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=3
	27:26	**Pixel Hashing Table Entry y[3]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=3
	25:24	**Pixel Hashing Table Entry y[3]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=3
	23:22	**Pixel Hashing Table Entry y[3]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=3
	21:20	**Pixel Hashing Table Entry y[3]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=3
	19:18	**Pixel Hashing Table Entry y[3]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=3
	17:16	**Pixel Hashing Table Entry y[3]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=3
	15:14	**Pixel Hashing Table Entry y[2]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=2
	13:12	**Pixel Hashing Table Entry y[2]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=2
	11:10	**Pixel Hashing Table Entry y[2]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=2

PIXEL_HASH_TABLE_2BIT_64ENTRY

	9:8	**Pixel Hashing Table Entry y[2]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=2
	7:6	**Pixel Hashing Table Entry y[2]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=2
	5:4	**Pixel Hashing Table Entry y[2]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=2
	3:2	**Pixel Hashing Table Entry y[2]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=2
	1:0	**Pixel Hashing Table Entry y[2]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=2
2	31:30	**Pixel Hashing Table Entry y[5]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=5
	29:28	**Pixel Hashing Table Entry y[5]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=5
	27:26	**Pixel Hashing Table Entry y[5]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=5
	25:24	**Pixel Hashing Table Entry y[5]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=5
	23:22	**Pixel Hashing Table Entry y[5]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=5

PIXEL_HASH_TABLE_2BIT_64ENTRY

	21:20	**Pixel Hashing Table Entry y[5]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=5
	19:18	**Pixel Hashing Table Entry y[5]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=5
	17:16	**Pixel Hashing Table Entry y[5]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=5
	15:14	**Pixel Hashing Table Entry y[4]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=4
	13:12	**Pixel Hashing Table Entry y[4]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=4
	11:10	**Pixel Hashing Table Entry y[4]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=4
	9:8	**Pixel Hashing Table Entry y[4]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=4
	7:6	**Pixel Hashing Table Entry y[4]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=4
	5:4	**Pixel Hashing Table Entry y[4]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=4
	3:2	**Pixel Hashing Table Entry y[4]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=4

PIXEL_HASH_TABLE_2BIT_64ENTRY

	1:0	**Pixel Hashing Table Entry y[4]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=4
3	31:30	**Pixel Hashing Table Entry y[7]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=7
	29:28	**Pixel Hashing Table Entry y[7]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=7
	27:26	**Pixel Hashing Table Entry y[7]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=7
	25:24	**Pixel Hashing Table Entry y[7]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=7
	23:22	**Pixel Hashing Table Entry y[7]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=7
	21:20	**Pixel Hashing Table Entry y[7]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=7
	19:18	**Pixel Hashing Table Entry y[7]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=7
	17:16	**Pixel Hashing Table Entry y[7]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=7
	15:14	**Pixel Hashing Table Entry y[6]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=6

PIXEL_HASH_TABLE_2BIT_64ENTRY		
	13:12	**Pixel Hashing Table Entry y[6]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=6
	11:10	**Pixel Hashing Table Entry y[6]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=6
	9:8	**Pixel Hashing Table Entry y[6]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=6
	7:6	**Pixel Hashing Table Entry y[6]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=6
	5:4	**Pixel Hashing Table Entry y[6]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=6
	3:2	**Pixel Hashing Table Entry y[6]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=6
	1:0	**Pixel Hashing Table Entry y[6]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=6

PIXEL_HASH_TABLE_2BIT_128ENTRY

PIXEL_HASH_TABLE_2BIT_128ENTRY

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description
3-wayor 4-way pixel hashing table. Table is 128-entries:8X,8Y in [Y][X] format. Each entry is two bits that indicates which sub-slice hardware block the indicated xy pixel block is mapped.
pixelhash_id maps to subslice. A value of 0 indicates the first enabled subslice. A value of 1 indicates the second enabled subslice.

DWord	Bit	Description
0	31:30	**Pixel Hashing Table Entry y[0]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=0
	29:28	**Pixel Hashing Table Entry y[0]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=0
	27:26	**Pixel Hashing Table Entry y[0]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=0
	25:24	**Pixel Hashing Table Entry y[0]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=0
	23:22	**Pixel Hashing Table Entry y[0]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=0
	21:20	**Pixel Hashing Table Entry y[0]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=0

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	19:18	**Pixel Hashing Table Entry y[0]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=0
	17:16	**Pixel Hashing Table Entry y[0]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=0
	15:14	**Pixel Hashing Table Entry y[0]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=0
	13:12	**Pixel Hashing Table Entry y[0]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=0
	11:10	**Pixel Hashing Table Entry y[0]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=0
	9:8	**Pixel Hashing Table Entry y[0]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=0
	7:6	**Pixel Hashing Table Entry y[0]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=0
	5:4	**Pixel Hashing Table Entry y[0]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=0
	3:2	**Pixel Hashing Table Entry y[0]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=0
	1:0	**Pixel Hashing Table Entry y[0]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=0

PIXEL_HASH_TABLE_2BIT_128ENTRY		
1	31:30	**Pixel Hashing Table Entry y[1]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=1
	29:28	**Pixel Hashing Table Entry y[1]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=1
	27:26	**Pixel Hashing Table Entry y[1]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=1
	25:24	**Pixel Hashing Table Entry y[1]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=1
	23:22	**Pixel Hashing Table Entry y[1]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=1
	21:20	**Pixel Hashing Table Entry y[1]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=1
	19:18	**Pixel Hashing Table Entry y[1]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=1
	17:16	**Pixel Hashing Table Entry y[1]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=1
	15:14	**Pixel Hashing Table Entry y[1]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=1
	13:12	**Pixel Hashing Table Entry y[1]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=1

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	11:10	**Pixel Hashing Table Entry y[1]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=1
	9:8	**Pixel Hashing Table Entry y[1]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=1
	7:6	**Pixel Hashing Table Entry y[1]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=1
	5:4	**Pixel Hashing Table Entry y[1]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=1
	3:2	**Pixel Hashing Table Entry y[1]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=1
	1:0	**Pixel Hashing Table Entry y[1]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=1
2	31:30	**Pixel Hashing Table Entry y[2]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=2
	29:28	**Pixel Hashing Table Entry y[2]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=2
	27:26	**Pixel Hashing Table Entry y[2]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=2
	25:24	**Pixel Hashing Table Entry y[2]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=2

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	23:22	**Pixel Hashing Table Entry y[2]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=2
	21:20	**Pixel Hashing Table Entry y[2]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=2
	19:18	**Pixel Hashing Table Entry y[2]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=2
	17:16	**Pixel Hashing Table Entry y[2]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=2
	15:14	**Pixel Hashing Table Entry y[2]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=2
	13:12	**Pixel Hashing Table Entry y[2]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=2
	11:10	**Pixel Hashing Table Entry y[2]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=2
	9:8	**Pixel Hashing Table Entry y[2]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=2
	7:6	**Pixel Hashing Table Entry y[2]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=2
	5:4	**Pixel Hashing Table Entry y[2]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=2

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	3:2	**Pixel Hashing Table Entry y[2]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=2
	1:0	**Pixel Hashing Table Entry y[2]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=2
3	31:30	**Pixel Hashing Table Entry y[3]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=3
	29:28	**Pixel Hashing Table Entry y[3]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=3
	27:26	**Pixel Hashing Table Entry y[3]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=3
	25:24	**Pixel Hashing Table Entry y[3]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=3
	23:22	**Pixel Hashing Table Entry y[3]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=3
	21:20	**Pixel Hashing Table Entry y[3]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=3
	19:18	**Pixel Hashing Table Entry y[3]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=3
	17:16	**Pixel Hashing Table Entry y[3]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=3

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	15:14	**Pixel Hashing Table Entry y[3]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=3
	13:12	**Pixel Hashing Table Entry y[3]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=3
	11:10	**Pixel Hashing Table Entry y[3]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=3
	9:8	**Pixel Hashing Table Entry y[3]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=3
	7:6	**Pixel Hashing Table Entry y[3]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=3
	5:4	**Pixel Hashing Table Entry y[3]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=3
	3:2	**Pixel Hashing Table Entry y[3]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=3
	1:0	**Pixel Hashing Table Entry y[3]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=3
4	31:30	**Pixel Hashing Table Entry y[4]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=4
	29:28	**Pixel Hashing Table Entry y[4]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=4

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	27:26	**Pixel Hashing Table Entry y[4]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=4
	25:24	**Pixel Hashing Table Entry y[4]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=4
	23:22	**Pixel Hashing Table Entry y[4]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=4
	21:20	**Pixel Hashing Table Entry y[4]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=4
	19:18	**Pixel Hashing Table Entry y[4]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=4
	17:16	**Pixel Hashing Table Entry y[4]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=4
	15:14	**Pixel Hashing Table Entry y[4]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=4
	13:12	**Pixel Hashing Table Entry y[4]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=4
	11:10	**Pixel Hashing Table Entry y[4]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=4
	9:8	**Pixel Hashing Table Entry y[4]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=4

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	7:6	**Pixel Hashing Table Entry y[4]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=4
	5:4	**Pixel Hashing Table Entry y[4]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=4
	3:2	**Pixel Hashing Table Entry y[4]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=4
	1:0	**Pixel Hashing Table Entry y[4]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=4
5	31:30	**Pixel Hashing Table Entry y[5]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=5
	29:28	**Pixel Hashing Table Entry y[5]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=5
	27:26	**Pixel Hashing Table Entry y[5]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=5
	25:24	**Pixel Hashing Table Entry y[5]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=5
	23:22	**Pixel Hashing Table Entry y[5]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=5
	21:20	**Pixel Hashing Table Entry y[5]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=5

PIXEL_HASH_TABLE_2BIT_128ENTRY

	19:18	**Pixel Hashing Table Entry y[5]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=5
	17:16	**Pixel Hashing Table Entry y[5]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=5
	15:14	**Pixel Hashing Table Entry y[5]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=5
	13:12	**Pixel Hashing Table Entry y[5]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=5
	11:10	**Pixel Hashing Table Entry y[5]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=5
	9:8	**Pixel Hashing Table Entry y[5]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=5
	7:6	**Pixel Hashing Table Entry y[5]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=5
	5:4	**Pixel Hashing Table Entry y[5]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=5
	3:2	**Pixel Hashing Table Entry y[5]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=5
	1:0	**Pixel Hashing Table Entry y[5]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=5

PIXEL_HASH_TABLE_2BIT_128ENTRY		
6	31:30	**Pixel Hashing Table Entry y[6]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=6
	29:28	**Pixel Hashing Table Entry y[6]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=6
	27:26	**Pixel Hashing Table Entry y[6]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=6
	25:24	**Pixel Hashing Table Entry y[6]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=6
	23:22	**Pixel Hashing Table Entry y[6]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=6
	21:20	**Pixel Hashing Table Entry y[6]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=6
	19:18	**Pixel Hashing Table Entry y[6]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=6
	17:16	**Pixel Hashing Table Entry y[6]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=6
	15:14	**Pixel Hashing Table Entry y[6]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=6
	13:12	**Pixel Hashing Table Entry y[6]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=6

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	11:10	**Pixel Hashing Table Entry y[6]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=6
	9:8	**Pixel Hashing Table Entry y[6]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=6
	7:6	**Pixel Hashing Table Entry y[6]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=6
	5:4	**Pixel Hashing Table Entry y[6]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=6
	3:2	**Pixel Hashing Table Entry y[6]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=6
	1:0	**Pixel Hashing Table Entry y[6]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=6
7	31:30	**Pixel Hashing Table Entry y[7]x[15]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=15 and y=7
	29:28	**Pixel Hashing Table Entry y[7]x[14]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=14 and y=7
	27:26	**Pixel Hashing Table Entry y[7]x[13]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=13 and y=7
	25:24	**Pixel Hashing Table Entry y[7]x[12]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=12 and y=7

PIXEL_HASH_TABLE_2BIT_128ENTRY

	23:22	**Pixel Hashing Table Entry y[7]x[11]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=11 and y=7
	21:20	**Pixel Hashing Table Entry y[7]x[10]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=10 and y=7
	19:18	**Pixel Hashing Table Entry y[7]x[9]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=9 and y=7
	17:16	**Pixel Hashing Table Entry y[7]x[8]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=8 and y=7
	15:14	**Pixel Hashing Table Entry y[7]x[7]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=7 and y=7
	13:12	**Pixel Hashing Table Entry y[7]x[6]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=6 and y=7
	11:10	**Pixel Hashing Table Entry y[7]x[5]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=5 and y=7
	9:8	**Pixel Hashing Table Entry y[7]x[4]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=4 and y=7
	7:6	**Pixel Hashing Table Entry y[7]x[3]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=3 and y=7
	5:4	**Pixel Hashing Table Entry y[7]x[2]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=2 and y=7

PIXEL_HASH_TABLE_2BIT_128ENTRY		
	3:2	**Pixel Hashing Table Entry y[7]x[1]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=1 and y=7
	1:0	**Pixel Hashing Table Entry y[7]x[0]** Format: U2 Indicates the pixelhash_id for the pixel block that has x=0 and y=7

Pixel Masked Media Block Message Header

MH_MBPM - Pixel Masked Media Block Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** When Message Mode is set to PIXEL_MASK, this field must be a multiple of 32.
1	31:0	**Y Offset** Format: S31 Y offset (in rows) of the upper left corner of the block into the surface. **Programming Notes** When Message Mode is set to PIXEL_MASK, this field must be a multiple of 4.
2	31:0	**Media Block Message Control** Format: MHC_MBPM_CONTROL Specifies the message subtype is Pixel Masked.
3	31:0	**Pixel Mask** Format: U32 Specifies the Pixel Mask for writes when Message Mode field is PIXEL_MASK. **Programming Notes** The Pixel Mask applies to the 2x2 square tiles (UL, UR, LL, LR), which themselves tiled (UL, UR, LL, LR) and then repeated on the right for the remaining 16-bits to cover a 4 row 8 column area.
4	31:0	**FFTID** Format: MHC_FFTID Fixed Function Thread ID
5..7	95:0	**Reserved** Format: Ignore Ignored

Pixel Masked Media Block Message Header Control

MHC_MBPM_CONTROL - Pixel Masked Media Block Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Format: Enumeration Specifies the Media Block Write Message subtype is Pixel Masked. Value / Name / Description: 01h / PIXEL_MASK / Use the Pixel Mask in the Message Header. The Block Height and Block Width are ignored and behave as if they are set to 4 rows and 32 bytes, respectively. Others / Reserved / Reserved.
	29	**Reserved** Format: Ignore Ignored
	28:24	**Sub-Register Offset** Format: MBZ The sub-register offset must be 0 for Pixel Masked Media Block message subtype.
	23:22	**Reserved** Format: Ignore Ignored
	21:16	**Block Height** Format: U6 This field is ignored (reserved) for a Pixel Masked media block write message.
	15:10	**Reserved** Format: Ignore Ignored
	9:8	**Register Pitch Control** Format: MBZ The register pitch must be 0 for a Pixel Masked Media Block message.

MHC_MBPM_CONTROL - Pixel Masked Media Block Message Header Control

	7:6	**Reserved** Format: Ignore Ignored
	5:0	**Block Width** Format: U6 This field is ignored (reserved) for a Pixel Masked media block write message.

Pixel Sample Mask Message Header Control

MHC_PSM - Pixel Sample Mask Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x0000FFFF

DWord	Bit	Description
0	31:16	**Reserved** Format: Ignore Ignored
	15:0	**Pixel Sample Mask** Default Value: 0FFFFh Default Format: U16 SIMD16 and SIMD8 messages. All 16 bits are used for SIMD16. For untyped SIMD8 messages, the low 8 bits of field are used. If the header is not delivered, this field defaults to all ones. This field is ignored for SIMD4x2 messages.

Pixel Sample Mask Render Target Message Header Control

MHC_RT_PSM - Pixel Sample Mask Render Target Message Header Control

Source:	BSpec
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Dispatched Pixel/Sample Enables** Format: U16 One bit per pixel (or sample within pixel) indicating which pixels/samples were originally enabled when the thread was dispatched. The Dispatched Pixel/Sample Enables must be unmodified from the ones sent when the pixel shader thread was initiated. If the Dispatched Pixel/Sample Enables are modified, behavior is undefined. **Programming Notes** When operating in PER_SAMPLE mode these bits correspond to samples, not pixels. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. Note that in NUMSAMPLES_1 mode, a pixel and sample are synonomous. When operating in PER_PIXEL mode, this field is ignored, and instead the SampleEnableMask (obtained via bypass) are used to clear the Depth Scoreboard.
	15:0	**Pixel/Sample Enables** Format: U16 Specifies which pixels/samples are still lit based on kill instruction activity in the pixel shader. This mask is AND'd with the Dispatched Pixel/Sample Enables mask, and that is used to control actual accesses to the color buffer. Pixels/samples will be dropped on masked writes, and the GRF is not modified for masked reads. **Programming Notes** When operating in PER_SAMPLE mode these bits correspond to samples, not pixels, as the PS is run per-sample. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. When operating in PER_PIXEL mode, these bits still correspond to pixels, as the PS is run per-pixel. Each pixel's mask bit is replicated according to Number of Multisamples and combined with other masks to control writes to the multisample locations.

Power Clock State Format

Power Clock State Format

Source: RenderCS
Size (in bits): 31
Default Value: 0x00000088

Known Uses

- R_PWR_CLK_STATE - Render Power Clock State Register
- PM_PWR_CLK_STATE - PM Power Clock State Request (Intended, in GT/GTI space, not yet in use)
- PM_PWR_CLK_STATE (Intended, in GT/GTI space, not yet in use)

DWord	Bit	Description
0	30:20	**Reserved** Access: RO Format: MBZ
	19	**Reserved** Access: RO Format: MBZ
	18	**Enable Slice Count Request** Access: R/W Enable Slice Count Request. Value / Name / Description 0h / Disable / Use async PMunit slice count request. 1h / Enable / Use SliceCount from this register.
	17:15	**Slice Count Request** Access: R/W Slice Count Request. This is limited to the number of slices in a given SKU. Value / Name / Description 001b / / 1 slice. 010b / / 2 slices. 011b / / 3 slices. 100b / / 4 slices. 101b / / 5 slices. Hardware will revert to 4 slices 110b / / 6 slices.

Power Clock State Format

	14:12	**Reserved** Access: RO Format: MBZ
	11:8	**Reserved** Access: RO Format: MBZ
	8	**Reserved** Access: RO
	7:4	**EUmax** Access: R/W Maximum number of EUs to power (per subslice if multiple subslices enabled). To specify an exact number of subslices, set EUmax equal to EUmin.

Value	Name	Description
0010b		2 EUs
0100b		4 EUs
0110b		6 EUs
1000b	**[Default]**	8 EUs

Programming Notes
EUmin and EUmax need to be even and odd numbers are illegal; hardware will clip odd EU counts to an even value.

	3:0	**EUmin** Access: R/W Minimum number of EUs to power (per subslice if multiple subslices enabled). To specify an exact number of subslices, set EUmax equal to EUmin.

Value	Name	Description
0010b		2 EUs
0100b		4 EUs
0110b		6 EUs
1000b	**[Default]**	8 EUs

Programming Notes
EUmin and EUmax need to be even and odd numbers are illegal; hardware will clip odd EU counts to an even value.

PPHWSP_LAYOUT

PPHWSP_LAYOUT - PPHWSP_LAYOUT	
Source:	BSpec
Size (in bits):	32641
Default Value:	0x00000000, 0x00000000, 0x00000000,

PPHWSP_LAYOUT - PPHWSP_LAYOUT

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

PPHWSP_LAYOUT - PPHWSP_LAYOUT

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

PPHWSP_LAYOUT - PPHWSP_LAYOUT

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

PPHWSP_LAYOUT - PPHWSP_LAYOUT

		The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
5..15	31:0	**Reserved**
16	0	**Cumulative Context Run Time** This has the cumulative run time of the context on HW. HW reports CTX_TIMESTAMP to this location on a context switch. This value is written after the context save is complete. The value that is saved in the context image does not include the time between the saving of the cumulative value to context to the time we complete the save. If required for the value to always increment and not take the context save into consideration, driver must look at the value in the context image.
17	31:1	**Reserved**
	0	**Element Switch** Value / Name / Description 0 / / Indicates the context is not submitted as the first element in the execlist. 1 / / Indicates the corresponding context has been submitted as first element of the execlist. Preempt Request Received Timestamp is the time when the pending execlist has been submitted to HW. Note that across multiple submissions a given context could be first or second element of an execlist. This bit will get set if the context has been submitted as the first element in the execlist.
18..19	63:0	**Preempt Request Received Timestamp** TIMESTAMP register sampled on preemption request is reported.
20..21	63:0	**Context Restore Complete Timestamp** TIMESTAMP register sampled on context restore complete is reported.
22..23	63:0	**Context Save Finished Timestamp** TIMESTAMP register sampled on context save completion is reported.
24..27	127:0	**MI_SEMAPHORE_WAIT** MI_SEMAPHORE_WAIT command on which the context got switched out due to semaphore wait. This field is only valid and must be looked at when the context switch reason in context status buffer is stated as "Wait on Semaphore".
28..31	127:0	**Reserved**
32..33	63:0	**Reserved**
34..1020	31:0	**Reserved**

Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload

MDP_A64_AOP4X2_QW2 - Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Src0 Slot0** Format: U64 Specifies the Slot 0 Source 0 data
0.2-0.3	63:0	**Reserved** Format: Ignore Ignored
0.4-0.5	63:0	**Src0 Slot1** Format: U64 Specifies the Slot 1 Source 0 data
0.6-0.7	63:0	**Reserved** Format: Ignore Ignored
1.0-1.1	63:0	**Src1 Slot0** Format: U64 Specifies the Slot 0 Source 1 data
1.2-1.3	63:0	**Reserved** Format: Ignore Ignored
1.4-1.5	63:0	**Src1 Slot1** Format: U64 Specifies the Slot 1 Source 1 data

MDP_A64_AOP4X2_QW2 - Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload		
1.6-1.7	63:0	**Reserved** Format: Ignore Ignored

Qword Data Payload Register

MDCR_QW - Qword Data Payload Register

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Qword0** Format: U64 Specifies the slot 0 data in this payload register
0.2-0.3	63:0	**Qword1** Format: U64 Specifies the slot 1 data in this payload register
0.4-0.5	63:0	**Qword2** Format: U64 Specifies the slot 2 data in this payload register
0.6-0.7	63:0	**Qword3** Format: U64 Specifies the slot 3 data in this payload register

Qword SIMD4x2 Atomic CMPWR8B Message Data Payload

MDP_AOP4X2_QW2 - Qword SIMD4x2 Atomic CMPWR8B Message Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**Src0 Slot0** Format: U64 Specifies the Slot 0 Source 0 data
2..3	63:0	**Src1 Slot0** Format: U64 Specifies the Slot 0 Source 1 data
4..5	63:0	**Src0 Slot1** Format: U64 Specifies the Slot 1 Source 0 data
6..7	63:0	**Src1 Slot1** Format: U64 Specifies the Slot 1 Source 1 data

Qword SIMD4x2 Atomic Operation Message Data Payload

MDP_AOP4X2_QW1 - Qword SIMD4x2 Atomic Operation Message Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**Qword0** Format: U64|S63 Specifies the Slot 0 Source or Return data
2..3	63:0	**Reserved** Format: Ignore Ignored
4..5	63:0	**Qword1** Format: U64|S63 Specifies the Slot 1 Source or Return data
6..7	63:0	**Reserved** Format: Ignore Ignored

Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload

MDP_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Src0[31:0]** Format: MDCR_DW Specifies the lower 32-bits of Slot [7:0] Source 0 data
1.0-1.7	255:0	**Slot[7:0] Src0[63:32]** Format: MDCR_DW Specifies the upper 32-bits of Slot [7:0] Source 0 data
2.0-2.7	255:0	**Slot[7:0] Src1[31:0]** Format: MDCR_DW Specifies the lower 32-bits of Slot [7:0] Source 1 data
3.0-3.7	255:0	**Slot[7:0] Src1[63:32]** Format: MDCR_DW Specifies the upper 32-bits of Slot [7:0] Source 1 data

Qword SIMD8 Atomic Operation CMPWR Message Data Payload

MDP_A64_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR Message Data Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[3:0] Src0** Format: MDCR_QW Specifies the Slot [3:0] Source 0 data
1.0-1.7	255:0	**Slot[7:4] Src0** Format: MDCR_QW Specifies the Slot [7:4] Source 0 data
2.0-2.7	255:0	**Slot[3:0] Src1** Format: MDCR_QW Specifies the Slot [3:0] Source 1 data
3.0-3.7	255:0	**Slot[7:4] Src1** Format: MDCR_QW Specifies the Slot [7:4] Source 1 data

Qword SIMD8 Atomic Operation Return Data Message Data Payload

MDP_AOP8_QW1 - Qword SIMD8 Atomic Operation Return Data Message Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Qword[31:0]** Format: MDCR_DW Specifies the lower 32-bits of Slot [7:0] Return data
1.0-1.7	255:0	**Slot[7:0] Qword[63:32]** Format: MDCR_DW Specifies the upper 32-bits of Slot [7:0] Return data

Qword SIMD8 Data Payload

MDP_QW_SIMD8 - Qword SIMD8 Data Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[3:0]** Format: MDCR_QW Specifies the Slot [3:0] data
1.0-1.7	255:0	**Data[7:4]** Format: MDCR_QW Specifies the Slot [7:4] data

Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload

MDP_AOP16_QW2 - Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload

Source:	BSpec
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_AOP16_QW2 - Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload		
5.0-5.7	255:0	**Slot[15:8] Src1[31:0]** Format: **MDCR_DW** Specifies the lower 32-bits Source 1 data for Slot [15:8]
6.0-6.7	255:0	**Slot[7:0] Src1[63:32]** Format: **MDCR_DW** Specifies the upper 32-bits of Source 1 data for Slot [7:0]
7.0-7.7	255:0	**Slot[15:8] Src1[63:32]** Format: **MDCR_DW** Specifies the upper 32-bits Source 1 data for Slot [15:8]

Qword SIMD16 Atomic Operation Return Data Message Data Payload

MDP_AOP16_QW1 - Qword SIMD16 Atomic Operation Return Data Message Data Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Qword[31:0]** Format: MDCR_DW Specifies the lower 32-bits of Return data for Slot [7:0]
1.0-1.7	255:0	**Slot[15:8] Qword[31:0]** Format: MDCR_DW Specifies the lower 32-bits of Return data for Slot [15:8]
2.0-2.7	255:0	**Slot[7:0] Qword[63:32]** Format: MDCR_DW Specifies the upper 32-bits of Return data for Slot [7:0]
3.0-3.7	255:0	**Slot[15:8] Qword[63:32]** Format: MDCR_DW Specifies the upper 32-bits of Return data for Slot [15:8]

Qword SIMD16 Data Payload

MDP_QW_SIMD16 - Qword SIMD16 Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[3:0]** Format: MDCR_QW Specifies the Slot [3:0] data
1.0-1.7	255:0	**Data[7:4]** Format: MDCR_QW Specifies the Slot [7:4] data
2.0-2.7	255:0	**qw11_qw8** Format: MDCR_QW Specifies the Slot [11:8] data
3.0-3.7	255:0	**qw15_qw12** Format: MDCR_QW Specifies the Slot [15:12] data

Read-Only Data Port Message Types

MT_DP_RO - Read-Only Data Port Message Types

Source: Read-Only DataPort

Size (in bits): 5

Default Value: 0x00000000

Lists all the Message Types in a Read-Only Data Port Message Descriptor [18:14].
Read operations from the Constant Cache and Sampler Cache are encoded in the Read-Only Data Port.
Many of the operations are also implemented in Data Port 0, and those operations use the same Message Header.

DWord	Bit	Description
0	4	**Reserved** Format: MBZ Ignored
	3:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description
00h	MT_CC_OWB **[Default]**	Oword Block Read Constant Cache message
01h	MT_CC_OWUB	Unaligned Oword Block Read Constant Cache message
02h	MT_CC_OWDB	Oword Dual Block Read Constant Cache message
03h	MT_CC_DWS	Dword Scattered Read Constant Cache message
04h	MT_SC_OWUB	Unaligned Oword Block Read Sampler Cache message
05h	MT_SC_MB	Media Block Read Sampler Cache message
06h	MT_RSI	Read Surface Info message
Others	Reserved	Ignored

Read Surface Info 32-Bit Address Payload

MAP32B_RSI - Read Surface Info 32-Bit Address Payload

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**U** Format: U32 Specifies the U channel address offset.
0.1	31:0	**V** Format: U32 Specifies the V channel address offset.
0.2	31:0	**R** Format: U32 Specifies the R channel address offset.
0.3	31:0	**LOD** Format: MACD_LOD Specifies the LOD.
0.4-0.7	127:0	**Reserved** Format: Ignore Ignored

Read Surface Info Data Payload

MDP_RSI - Read Surface Info Data Payload

Source: BSpec

Size (in bits): 512

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.5	191:0	**Reserved** Format: Ignore Ignored
0.6-0.7	63:0	**Instruction Base Address** Format: GraphicsAddress[63:0] Instruction Base Address from STATE_BASE_ADDRESS, extended to 64-bit format. **Programming Notes** The 48-bit address is returned in a 64-bit address in canonical form.
1.0	31:0	**Width** Format: U32 Surface Width generally computed from RENDER_SURFACE_STATE Width (stored as width minus 1). The value is 0 for NULL surface, and in all other cases (Width+1) » LOD. Surface Width from RENDER_SURFACE_STATE (U14), zero extended to 32 bits.
1.1	31:0	**Height** Format: U32 Surface Height, generally computed from RENDER_SURFACE_STATE Height (stored as height minus 1). The value for a 1D array is RENDER_SUFACE_STATE's (Depth + 1). The value for 1D non-array, BUFFER, and NULL surface is 0. In all other case, the value is (Height + 1) » LOD.
1.2	31:0	**Depth** Format: U32 Surface Depth, generally computed from RENDER_SURFACE_STATE Depth (which is stored depth minus 1). If 2D Array or Cube Array surface, value is the (Depth+1). If 3D surface, value is (Depth+1) » LOD. In all other case, the value is 0.
1.3	31:0	**MIP Count** Format: U32 MIP Count from RENDER_SURFACE_STATE, range [0, 14], zero extended to 32 bits.

MDP_RSI - Read Surface Info Data Payload

1.4 31:0 Surface Type

Format:	U32

Surface Type from RENDER_SURFACE_STATE, zero extended to 32 bits

Value	Name	Description
0h	SURFTYPE_1D	1-dimensional map or array of maps
1h	SURFTYPE_2D	2-dimensional map or array of maps
2h	SURFTYPE_3D	3-dimensional map (volumetric) of maps
3h	SURFTYPE_CUBE	Cube map or array of cube maps
4h	SURFTYPE_BUFFER	Element in a buffer
5h	SURFTYPE_STRBUF	Structured buffer surface
7h	SURTYPE_NULL	Null surface
Others	Reserved	Reserved

1.5 31:0 Surface Format

Format:	U32

Surface Format from RENDER_SURFACE_STATE (U9), zero extended to 32 bits.

1.6-1.7 63:0 Reserved

Format:	Ignore

Ignored

RENDER_SURFACE_STATE

RENDER_SURFACE_STATE

Source:	BSpec
Exists If:	//[MessageType] != 'Sample_8x8'
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the normal surface state used by all messages that use SURFACE_STATE except those that use MEDIA_SURFACE_STATE.

DWord	Bit	Description
0	31:29	**Surface Type** This field defines the type of the surface.

Value	Name	Description
0h	SURFTYPE_1D	Defines a 1-dimensional map or array of maps
1h	SURFTYPE_2D	Defines a 2-dimensional map or array of maps
2h	SURFTYPE_3D	Defines a 3-dimensional (volumetric) map
3h	SURFTYPE_CUBE	Defines a cube map or array of cube maps
4h	SURFTYPE_BUFFER	Defines an element in a buffer
5h	SURFTYPE_STRBUF	Defines a structured buffer surface
6h	Reserved	
7h	SURFTYPE_NULL	Defines a null surface

Programming Notes

A null surface is used in instances where an actual surface is not bound. When a write message is generated to a null surface, no actual surface is written to. When a read message (including any sampling engine message) is generated to a null surface, the result is all zeros. Note that a null surface type is allowed to be used with all messages, even if it is not specificially indicated as supported. All of the remaining fields in surface state are ignored for null surfaces, with the following exceptions:

- **Width, Height, Depth, LOD,** and **Render Target View Extent** fields must match the depth buffer's corresponding state for all render target surfaces, including null.

All sampling engine and data port messages support null surfaces with the above behavior, even if not mentioned as specifically supported, except for the following:

- The **Surface Type** of a surface used as a render target (accessed via the Data Port's Render Target Write message) must be the same as the **Surface Type** of all other render targets and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless either the depth buffer or render targets are SURFTYPE_NULL.

RENDER_SURFACE_STATE

	28	**Surface Array** Format: Enable This field, if enabled, indicates that the surface is an array. **Programming Notes** If this field is *enabled*, the **Surface Type** must be SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE. If this field is *disabled* and **Surface Type** is SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE, the **Depth** field must be set to zero.
	27	**ASTC_Enable** Format: Enable This field, if enabled, indicates that the surface is one of ASTC compression formats. **Programming Notes** If this field is *enabled*, the definition of **Surface Format** encoding will follow a new convention defined by ASTC. If this field is *disabled*, the definition of **Surface Format** will follow the legacy convention defined in non-ASTC style.
	26:18	**Surface Format** Format: SURFACE_FORMAT **Description** This field specifies the format of the surface or element within this surface. This field is ignored for all data port messages other than the render target message and streamed vertex buffer write message. Some forms of the media block messages use the surface format. If **ASTC_Enable** is set to 0, the supported formats and their encoding is listed in the table (x) in Section (y); Otherwise the supported formats and their encoding is listed in the table (x+1) in Section (y). **Programming Notes** If **ASTC_Enable** is set to 0: YUV (YCRCB) surfaces used as render targets can only be rendered to using 3DPRIM_RECTLIST with even X coordinates on all of its vertices, and the pixel shader cannot kill pixels. If **Number of Multisamples** is set to a value other than MULTISAMPLECOUNT_1, this field cannot be set to the following formats: • Any compressed texture format (BC*, DXT*, FXT*, ETC*, EAC*) • Any YCRCB* format This field cannot ASTC format if the **Surface Type** is SURFTYPE_BUFFER or SURFTYPE_STRBUF This field cannot be ASTC format if the **Surface Type** is SURFTYPE_1D. This field cannot be a YUV (YCRCB*) or compressed (BC*, DXT*, FXT*, ETC*, EAC*) format if the **Surface Type** is SURFTYPE_BUFFER or SURFTYPE_STRBUF This field cannot be a planar YUV (PLANAR_*) or compressed (BC*, DXT*, FXT*, ETC*, EAC*) format if the **Surface Type** is SURFTYPE_1D.

RENDER_SURFACE_STATE

17:16 **Surface Vertical Alignment**

Description
For Sampling Engine and Render Target Surfaces: This field specifies the vertical alignment requirement in elements for the surface. Refer to the "Memory Data Formats" chapter for details on how this field changes the layout of the surface in memory. An *element* is defined as a pixel in uncompresed surface formats, and as a compression block in compressed surface formats. For MSFMT_DEPTH_STENCIL type multisampled surfaces, an element is a sample.
This field is used for 2D, CUBE, and 3D surface alignment when Tiled Resource Mode is TRMODE_NONE (Tiled Resource Mode is disabled). This field is ignored for 1D surfaces and also when Tiled Resource Mode is not TRMODE_NONE (e.g. Tiled Resource Mode is enabled). See the appropriate Alignment table in the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tiled Resrouces.
For other surfaces: This field is ignored.

Value	Name	Description
0h	Reserved	Reserved
1h	VALIGN 4	Vertical alignment factor j = 4
2h	VALIGN 8	Vertical alignment factor j = 8
3h	VALIGN 16	Vertical alignment factor j = 16

Programming Notes
This field is intended to be set to VALIGN_4 if the surface was rendered as a depth buffer, for a multisampled (4x) render target, or for a multisampled (8x) render target, since these surfaces support only alignment of 4. Use of VALIGN_4 for other surfaces is supported, but increases memory usage.
This field is intended to be set to VALIGN_8 only if the surface was rendered as a stencil buffer, since stencil buffer surfaces support only alignment of 8. If set to VALIGN_8, Surface Format must be R8_UINT.
For uncompressed surfaces, the units of "j" are rows of pixels on the physical surface. For compressed texture formats, the units of "j" are in compression blocks, thus each increment in "j" is equal to h pixels, where h is the height of the compression block in pixels.

15:14 **Surface Horizontal Alignment**

Description
For Sampling Engine and Render Target Surfaces: This field specifies the horizontal alignment requirement for the surface.
This field is used for alignment when LOD >= Mip Tail Start LOD This field is ignored when Tiled Resource Mode is not TRMODE_NONE (i.e. Tiled Resources are enabled). See the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile Resrouces.

RENDER_SURFACE_STATE

For other surfaces:
This field is ignored.

Value	Name	Description
0h	Reserved	Reserved
1h	HALIGN 4	Horizontal alignment factor j = 4
2h	HALIGN 8	Horizontal alignment factor j = 8
3h	HALIGN 16	Horizontal alignment factor j = 16

Programming Notes

This field is intended to be set to HALIGN_8 only if the surface was rendered as a depth buffer with Z16 format or a stencil buffer. In this case it must be set to HALIGN_8 since these surfaces support only alignment of 8. For Z32 formats it must be set ot HALIGN_4. Use of HALIGN_8 for other surfaces is supported, but increases memory usage.
For uncompressed surfaces, the units of "i" are pixels on the physical surface. For compressed texture formats, the units of "i" are in compression blocks, thus each increment in "i" is equal to w pixels, where w is the width of the compression block in pixels.

When Auxiliary Surface Mode is set to AUX_CCS_D or AUX_CCS_E, HALIGN 16 must be used.

13:12 **Tile Mode**
This field specifies the type of memory tiling (Linear, WMajor, XMajor, or YMajor) employed to tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions.

Value	Name	Description
0h	LINEAR	Linear mode (no tiling)
1h	WMAJOR	W major tiling
2h	XMAJOR	X major tiling
3h	YMAJOR	Y major tiling

Programming Notes

- Refer to *Memory Data Formats* for restrictions on *TileMode* direction for the various buffer types. (Of particular interest is the fact that YMAJOR tiling is not supported for display/overlay buffers).
- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this field.
- Use of WMAJOR is valid only for sampling engine, Data Cache Data Port and render target surfaces and **Surface Format** must be R8_UINT. Vertical Line Stride must be zero. In addition to W tiling, this mode implies that the surface is stored as a stencil buffer. Refer to *Memory Data Formats* section for details on stencil buffer surface layout.
- Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled (X/Y/W) surfaces can only be mapped to Main Memory.
- If **Surface Type** is SURFTYPE_BUFFER, this field must be TILEMODE_LINEAR

RENDER_SURFACE_STATE

- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be YMAJOR.

If **Surface Type** is SURFTYPE_STRBUF, this field must be TILEMODE_LINEAR.

If **Surface Type** is SURFTYPE_1D this field must be TILEMODE_LINEAR, unless **Sampler Legacy 1D Map Layout Disable** is set to 0, in which case TILEMODE_YMAJOR and TILEMODE_WMAJOR are also allowed. **Tiled Resource Mode** must be set to TRMODE_NONE for these cases.

TILEMODE_XMAJOR is only allowed if Surface Type is SURFTYPE_2D.

If **Surface Format** is ASTC*, this field must be TILEMODE_YMAJOR.

11 **Vertical Line Stride**

Format:	U1 In lines to skip between logically adjacent lines

For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Cache Data Port: Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.
For Other Surfaces: Vertical Line Stride must be zero.

Programming Notes

This bit must not be set if the surface format is a compressed type (BCn*, FXT1, ETC*, EAC*).

This bit must not be set if the surface format is compressed type ASTC*.

This bit must not be set if the **Auxiliary Surface Mode** is not AUX_NONE.

If this bit is set on a sampling engine surface, the mip mode filter must be set to MIPFILTER_NONE and the min and mag mode filter cannot be set to MAPFILTER_FLEXIBLE.

10 **Vertical Line Stride Offset**

Format:	U1 In lines of initial offset (when Vertical Line Stride == 1)

For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Cache Data Port: Specifies the offset of the initial line from the beginning of the buffer. Ignored when Vertical **Line Stride** is 0.
For Other Surfaces:
Vertical Line Stride Offset must be zero.

9 **Sampler L2 out of order mode Disable**

Format:	Disable

If disabled this will forced formats which would have bypassed the L2 and been filled into the L1 out of order to be cached in the L2 and send in order to the L1. In general that is any format which is expaned 1:2 in L1 or not expanded at all. This would include all lossless compressed cases
For all other formats this will have no affect.

8 **Render Cache Read Write Mode**
For Surfaces accessed via the Data Port to Render Cache:
This field specifies the way Render Cache treats a write request. If unset, Render Cache allocates a write-only cache line for a write miss. If set, Render Cache allocates a read-write cache line for a

RENDER_SURFACE_STATE

write miss.

For Surfaces accessed via the Sampling Engine or Data Port to Texture Cache or Data Cache:

This field is reserved : MBZ

Value	Name	Description
0h	Write-Only Cache	Allocating write-only cache for a write miss
1h	Read-Write Cache	Allocating read-write cache for a write miss

Programming Notes
This field is provided for performance optimization for Render Cache read/write accesses (from Gen4 EU's point of view).

7:6 **Media Boundary Pixel Mode**

For 2D Non-Array Surfaces accessed via the Data Port Media Block Read Message or Data Port Transpose Read message:

This field enables control of which rows are returned on vertical out-of-bounds reads using the Data Port Media Block Read Message or Data Port Transpose Read message. In the description below, frame mode refers to **Vertical Line Stride** = 0, field mode is **Vertical Line Stride** = 1 in which only the even or odd rows are addressable. The frame refers to the entire surface, while the field refers only to the even or odd rows within the surface.

For Other Surfaces:

Reserved : MBZ

Value	Name	Description
0h	NORMAL_MODE	The row returned on an out-of-bound access is the closest row in the frame or field. Rows from the opposite field are never returned.
1h	Reserved	
2h	PROGRESSIVE_FRAME	The row returned on an out-of-bound access is the closest row in the frame, even if in field mode.
3h	INTERLACED_FRAME	In field mode, the row returned on an out-of-bound access is the closest row in the field. In frame mode, even out-of-bound rows return the nearest even row while odd out-of-bound rows return the nearest odd row.

5 **Cube Face Enable - Negative X**

Exists If:	[Surface Type] == 'SURFTYPE_CUBE'
Format:	Enable

For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:

This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

RENDER_SURFACE_STATE

		When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	4	**Cube Face Enable - Positive X** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	3	**Cube Face Enable - Negative Y** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	2	**Cube Face Enable - Positive Y** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	1	**Cube Face Enable - Negative Z** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is

RENDER_SURFACE_STATE

present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).

5:0 Reserved

Exists If:	[Surface Type] != 'SURFTYPE_CUBE'
Format:	MBZ

0 Cube Face Enable - Positive Z

Exists If:	[Surface Type] == 'SURFTYPE_CUBE'
Format:	Enable

For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:

This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).

1 31 Reserved

Format:	MBZ

30:24 Memory Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).

23:19 Base Mip Level

Format:	U4.1

Range: [0.0, 14.0]

Specifies which mip level is considered the "base" level when determining mag-vs-min filter and selecting the "base" mip level.

Programming Notes

This field also exists in SAMPLER_STATE. If both fields are zero, the Base Mip Level is zero. If one is nonzero, Base Mip Level is the nonzero field. It is illegal to have both Base Mip Level fields nonzero.

RENDER_SURFACE_STATE

	18	**Reserved** Format: MBZ
	17	**Reserved** Format: MBZ
	16	**Reserved** Format: MBZ
	15	**Reserved** Format: MBZ
	14:0	**Surface QPitch** Format: QPitch[16:2]

Description

The interpretation of this field is dependent on Surface Type as follows:

- SURFTYPE_1D: distance in *pixels* between array slices
- SURFTYPE_2D/CUBE: distance in *rows* between array slices. For Quilted Textures this field specifies the distance in rows between *quilt* slices. For compressed texture formats, one row contains a complete compression block vertically.
- SURFTYPE_3D: distance in *rows* between R-slices [**Note:** these *rows* are only in the vertical dimension without considering the depth dimension]. For compressed texture formats, one row contains a complete compression block vertically.
- Other surface types: field is ignored

Value	Name	Description
[4h,1FFFCh]		in multiples of 4 (low 2 bits missing)

Programming Notes

For Surface Type 1D: This field must be set to an integer multiple of the **Surface Horizontal Alignment**
For Surface Type 2D, CUBE: This field must be set to an integer multiple of the **Surface Vertical Alignment**
For Surface Type 3D: *Tile Mode != Linear:* This field must be set to an integer multiple of the tile height (2^Cv) *Tile Mode == Linear:* This field must be set to an integer multiple of the Surface Vertical Alignment
Note: for compressed textures (BC*, FXT1, ETC*, EAC*), this field is in units of rows of compression blocks.

Note: for the compressed texture ASTC Surface Format, this field is in units of rows of compression blocks.

Software must ensure that this field is set to a value sufficiently large such that the array slices in the surface do not overlap. Refer to the Memory Data Formats section for information on

RENDER_SURFACE_STATE

how surfaces are stored in memory.

2 | 31:30 **Reserved**

Format:	MBZ

29:16 **Height**

Format:	U14-1

This field specifies the height of the surface, minus 1. If the surface is MIP-mapped, this field contains the height of the base MIP level. For buffers, this field specifies a portion of the buffer size.

Value	Name	Description	Exists If
[0,0]		must be zero	[Surface Type] == 'SURFTYPE_1D'
[0,16383]		height of surface - 1 (y/v dimension)	[SurfaceType] == 'SURFTYPE_2D'
[0,16383]		height of surface - 1 (y/v dimension)	[SurfaceType] == 'SURFTYPE_3D'
[0,16383]		height of surface - 1 (y/v dimension)	[SurfaceType] == 'SURFTYPE_CUBE'
[0,16383]		contains bits [20:7] of the number of entries in the buffer - 1	([SurfaceType] == 'SURFTYPE_BUFFER') || ([SurfaceType] == 'SURFTYPE_STRBUF')

Programming Notes

For typed buffer and structured buffer surfaces, the number of entries in the buffer ranges from 1 to 2^{27}. For raw buffer surfaces, the number of entries in the buffer is the number of bytes which can range from 1 to 2^{30}. After subtracting one from the number of entries, software must place the fields of the resulting 27-bit value into the **Height, Width**, and **Depth** fields as indicated, right-justified in each field. Unused upper bits must be set to zero.

If **Vertical Line Stride** is 1, this field indicates the height of the field, not the height of the frame

The **Height** of a render target must be the same as the **Height** of the other render targets and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped).

If this surface in memory is accessed with Vertical Line Stride set to both 0 and 1, this field must be an even value when Vertical Line Stride is 0.

If Media Pixel Boundary Mode is not set to NORMAL_MODE, this field must be an even value.

If Surface Format is PLANAR*, see **Planar Memory Organization** section for restrictions on the value of this field.

15:14 **Reserved**

Format:	MBZ

RENDER_SURFACE_STATE

13:0 **Width**

Format:	U14-1

Description
This field specifies the width of the surface, minus 1. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels. For buffers, this field specifies a portion of the buffer size.
For surfaces accessed with the Media Block Read/Write message, this field is in units of DWords.
For surfaces accessed with the Transpose Read Message, this field is in units of DWords.

Value	Name	Description	Exists If
[0,16383]		width of surface - 1 (x/u dimension)	[SurfaceType] == 'SURFTYPE_1D'
[0,16383]		width of surface - 1 (x/u dimension)	[SurfaceType] == 'SURFTYPE_2D'
[0,16383]		width of surface - 1 (x/u dimension)	[SurfaceType] == 'SURFTYPE_3D'
[0,16383]		width of surface - 1 (x/u dimension)	[SurfaceType] == 'SURFTYPE_CUBE'
[0,127]		contains bits [6:0] of the number of entries in the buffer - 1	([SurfaceType] == 'SURFTYPE_BUFFER') || ([SurfaceType] == 'SURFTYPE_STRBUF')

Programming Notes

- For surface types other than SURFTYPE_BUFFER or STRBUF The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field).
- For cube maps, Width must be set equal to the Height.
- For MONO8 textures, Width must be a multiple of 32 texels.
- The **Width** of a render target must be the same as the **Width** of the other render target(s) and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped).
- The **Width** of a render target with YUV surface format must be a multiple of 2.
- For SURFTYPE_BUFFER: The low two bits of this field must be 11 if the Surface Format is RAW (the size of the buffer must be a multiple of 4 bytes).

If **Surface Format** is PLANAR*, this field must be a multiple of 2

RENDER_SURFACE_STATE

3 — 31:21 Depth

Format:	U11-1

This field specifies the total number of levels, minus 1, for a volume texture or the number of array elements, minus 1, allowed to be accessed starting at the **Minimum Array Element** for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level. For buffers, this field specifies a portion of the buffer size.

Value	Name	Description	Exists If
[0,2047]		number of array elements - 1	[SurfaceType] == 'SURFTYPE_1D'
[0,2047]		number of array elements - 1	[SurfaceType] == 'SURFTYPE_2D'
[0,2047]		depth of surface - 1 (z/r dimension)	[SurfaceType] == 'SURFTYPE_3D'
[0,340]		number of array elements - 1 [see programming notes for range]	[SurfaceType] == 'SURFTYPE_CUBE'
[0,2047]		contains bits [31:21] of the number of entries in the buffer - 1	([SurfaceType] == SURFTYPE_BUFFER) OR ([SurfaceType] == 'SURFTYPE_STRBUF')

Programming Notes

The **Depth** of a render target must be the same as the **Depth** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER).

For SURFTYPE_CUBE: For Sampling Engine Surfaces and Typed Data Port Surfaces, the range of this field is [0,340], indicating the number of cube array elements (equal to the number of underlying 2D array elements divided by 6). For other surfaces, this field must be zero.

For SURFTYPE_1D, 2D, and CUBE: The range of this field is reduced by one for each increase from zero of **Minimum Array Element**. For example, if **Minimum Array Element** is set to 1024 on a 2D surface, the range of this field is reduced to [0,1023].

20 — Reserved

Format:	MBZ

19 — Reserved

Format:	MBZ

18 — Reserved

Format:	MBZ

17:0 — Surface Pitch

Format:	U18-1 Pitch in #Bytes

Surface Pitch Range:

- For surfaces of type SURFTYPE_BUFFER: [0,2047] -> [1B, 2048B]
- For surfaces of type SURFTYPE_STRBUF: [0,2047] -> [1B, 2048B]
- For other linear surfaces: [0, 262143] -> [1B, 256KB]

RENDER_SURFACE_STATE

- For X-tiled surface: [511, 262143] -> [512B, 256KB] = [1 tile, 512 tiles]
- For Y-tiled surfaces: [127, 262143]->[128B, 256KB] = [1 tile, 2048 tiles]
- For W-tiled surfaces: [127, 262143]->[128B, 256KB] = [1 tile, 2048 tiles]
- For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to *Memory Data Formats* section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1,262143] -> [(2^{Cu})B,256KB] = [1 tile, 256KB/(2^{Cu}) tiles]

This field specifies the surface pitch in (#Bytes - 1).
For surfaces of type SURFTYPE_BUFFER and SURFTYPE_STRBUF, this field indicates the size of the structure.

Programming Notes

- For linear *render target* surfaces and surfaces accessed with the typed data port messages, the pitch must be a multiple of the element size for non-YUV surface formats. Pitch must be a multiple of 2 * element size for YUV surface formats.
- For untyped data port messages, which are only supported with **Surface Type** SURFTYPE_BUFFER, the pitch is ignored and assumed to be 1 byte.
- For linear surfaces with **Surface Type** of SURFTYPE_STRBUF, the pitch must be a multiple of 4 bytes.
- For linear surfaces with **Surface Type** of SURFTYPE_BUFFER and **Surface Format** RAW, the pitch must be 1 byte.
- For other linear surfaces, the pitch can be any multiple of bytes.
- For tiled surfaces, the pitch must be a multiple of the tile width.

If the surface is a stencil buffer (and thus has **Tile Mode** set to TILEMODE_WMAJOR), the pitch must be set to 2x the value computed based on width, as the stencil buffer is stored with two rows interleaved. For details on the separate stencil buffer storage format in memory, see GPU Overview (vol1a), Memory Data Formats, Surface Layout, 2D Surfaces, Stencil Buffer Layout (section 8.20.4.8).

- The width of a tile depends on the surface format if Tiled Resource Enable is enabled. Refer to the Tiled Resource Enable field to determine which sub-mode applies to the surface format in use, and determine the Cu parameter from the Surface Layout section. The tile width is equal to 2^Cu bytes.
- For surfaces of type SURFTYPE_1D, this field is ignored.

The following table indicates the maximum byte width, frame width, and pitch size allowed when memory compression is on.

Tiling Mode	Pixel Format	Max Frame Width (bytes)	Max Frame Width (pixels)	Max Pitch (bytes)
Legacy 4K	8bpp	16k	16k	16k + 127
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127

RENDER_SURFACE_STATE

		64bpp	16k	2k	16k + 127
		128bpp	16k	1k	16k + 127
	TileYF	8bpp	8k	8k	8k + 63
		16bpp	16k	8k	16k + 127
		32bpp	16k	4k	16k + 127
		64bpp	16k	2k	16k + 255
		128bpp	16k	1k	16k + 255
	TileYS	8bpp	16k	16k	16k + 255
		16bpp	16k	8k	16k + 511
		32bpp	16k	4k	16k + 511
		64bpp	16k	2k	16k + 1023
		128bpp	16k	1k	16k + 1023

4 — 31 — Reserved

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	MBZ

4 — 30:29 — Render Target And Sample Unorm Rotation

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

Description

For Render Target Surfaces:
This field specifies the rotation of this render target surface when being written to memory.

For sample_unorm Messages: This field specifies the rotation of the data returned by sampler for sample_unorm message.

For Other Surfaces:
This field is ignored.

Value	Name	Description
0h	0DEG	No rotation (0 degrees)
1h	90DEG	Rotate by 90 degrees
2h	180DEG	Rotate by 180 degrees [for sample_unorm message]
3h	270DEG	Rotate by 270 degrees

Programming Notes

Programming Notes for Render Target Surfaces only

- Rotation is not supported for render targets of any type other than simple, non-mip-mapped, non-array 2D surfaces. The surface must be using tiled with X major.
- **Width** and **Height** fields apply to the dimensions of the surface before rotation.

RENDER_SURFACE_STATE

- For 90 and 270 degree rotated surfaces, the **Height** (rather than the **Width**) must be less than or equal to the **Surface Pitch** (specified in bytes).
- For 90 and 270 degree rotated surfaces, the actual **Height** and **Width** of the surface in pixels (not the field value which is decremented) must both be even.

Rotation is supported only for surfaces with the following surface formats: B5G6R5_UNORM, B5G6R5_UNORM_SRGB, R8G8B8A8_UNORM, R8G8B8A8_UNORM_SRGB, B8G8R8[A|X]8_UNORM, B8G8R8[A|X]8_UNORM_SRGB, B10G10R10[A|X]2_UNORM, B10G10R10A2_UNORM_SRGB, R10G10B10A2_UNORM, R10G10B10A2_UNORM_SRGB, R16G16B16A16_FLOAT, R16G16B16X16_FLOAT

28:18 **Minimum Array Element**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	U11

Description

17:7 **Render Target View Extent**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	U11-1

Range [0,2047] to indicate extent of [1,2048]
For Render Target and Typed Dataport 3D Surfaces:
This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to.
For Render Target and Typed Dataport 1D and 2D Surfaces:
This field must be set to the same value as the Depth field.
For Other Surfaces:
This field is ignored.

6 **Multisampled Surface Storage Format**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

This field indicates the storage format of the multisampled surface.

Value	Name	Description
0h	MSS	Multsampled surface was/is rendered as a render target
1h	DEPTH_STENCIL	Multisampled surface was rendered as a depth or stencil buffer

Programming Notes

- All multisampled render target surfaces must have this field set to MSFMT_MSS
- IF this field is MSFMT_DEPTH_STENCIL, the only sampling engine messages allowed are

RENDER_SURFACE_STATE

"ld2dms", "resinfo", and "sampleinfo".

- This field is ignored if **Number of Multisamples** is MULTISAMPLECOUNT_1

5:3 Number of Multisamples

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

This field indicates the number of multisamples on the surface.

Value	Name
0h	MULTISAMPLECOUNT_1
1h	MULTISAMPLECOUNT_2
2h	MULTISAMPLECOUNT_4
3h	MULTISAMPLECOUNT_8
4h	MULTISAMPLECOUNT_16
5h-7h	Reserved

Programming Notes

If this field is any value other than MULTISAMPLECOUNT_1, the **Surface Type** must be SURFTYPE_2D

This field must be set to MULTISAMPLECOUNT_1 unless the surface is a Sampling Engine surface or Render Target surface.

31:0 Reserved

Exists If:	[Surface Type] == 'SURFTYPE_STRBUF'
Format:	MBZ

2:0 Multisample Position Palette Index

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

This field indicates the index into the sample position palette that the multisampled surface is using. This field is only used as a return value for the sampleinfo message, and is otherwise not used by hardware.

Value	Name
[0,7]	

5 31:25 X Offset

Format:	PixelOffset[8:2]

This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the surface.

This field effectively loosens the alignment restrictions on the origin of tiled surfaces. Previously, tiled surface origin was (by definition) located at the base address, and thus needed to satisfy the 4KB base address alignment restriction. Now the origin can be specified at a finer (4-wide x 4-high pixel) resolution.

Value	Name	Description
[0,508]		In multiples of 4 (low 2 bits missing)

RENDER_SURFACE_STATE

Programming Notes

- For linear surfaces, this field must be zero.
- For surfaces accessed with the *Data Port Media Block Read/Write* message, the pixel size is assumed to be 32 bits in width.
- For surfaces accessed with the **Data Port Transpose Read message**, the pixel size is assumed to be 32 bits in width.
- For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero.
- If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero.
- If **Surface Type** not SURFTYPE_2D, this field must be zero.
- If **MIP Count** is not zero, this field must be zero.
- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be zero.
- If **Surface Array** is enabled, this field must be zero.
- If **Auxiliary Surface Mode** is not AUX_NONE, this field must be zero.
- If **Surface Vertical Alignment** is VALIGN_8, this field must be a multiple of 8.
- For **Surface Format** with 8 bits per element, this field must be a multiple of 16.
- For **Surface Format** with 16 bits per element, this field must be a multiple of 8.

- If **Tiled Resource Mode** is not TRMODE_NONE, this field must be zero.

24 **Reserved**

Format:	MBZ

23:21 **Y Offset**

Format:	RowOffset[4:2]

This field specifies the vertical offset in rows from the **Surface Base Address** to the start of the surface. (See additional description in the **X Offset** field.)

Value	Name	Description
[0,28]		In multiples of 4 (low two bits missing)

Programming Notes

- For linear surfaces, this field must be zero.
- For render targets in which the **Render Target Array Index** is not zero, this field must be zero.
- For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero.
- If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero.
- If **Surface Type** not SURFTYPE_2D, this field must be zero.

RENDER_SURFACE_STATE

- If **MIP Count** is not zero, this field must be zero.
- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be zero.
- If **Surface Array** is enabled, this field must be zero.
- If **Auxiliary Surface Mode** is not AUX_NONE, this field must be zero.

- If **Tiled Resource Mode** is not TRMODE_NONE, this field must be zero.

This field must be zero if Surface Format is Planar and the U and V planes are half-pitch (e.g. YV12 format).

20 **EWA Disable For Cube**

Format:	Disable

Specifies if EWA mode for LOD quality improvement needs to be disabled for cube maps.

Value	Name	Description
0h	Enable **[Default]**	EWA is enabled for cube maps
1h	Disable	EWA is disabled for cube maps

Programming Notes

This field indicates if EWA mode for LOD quality improvement needs to be disabled for cube maps.
By default EWA would be on for cube maps hence this field must be 0. If there is any spec violation seen with EWA on cube maps then this field must be set to 1 to disable EWA for cubes.

19:18 **Tiled Resource Mode**

For Sampling Engine, Render Target, and Typed/Untyped Surfaces:
This field specifies the tiled resource mode.

For other surfaces:
This field is ignored.

Value	Name	Description	Exists If
0h	NONE	No tiled resource	
1h	4KB	4KB tiled resources	[SurfaceType] == 'SURFTYPE_1D'
2h	64KB	64KB tiled resources	[SurfaceType] == 'SURFTYPE_1D'
1h	TILEYF	4KB tiled resources	[SurfaceType] != 'SURFTYPE_1D'
2h	TILEYS	64KB tiled resources	[SurfaceType] != 'SURFTYPE_1D'
3h	Reserved		

Programming Notes

If **Tile Mode** is not set to TILEMODE_YMAJOR, this field must be set to TRMODE_NONE, unless the Surface Type is SURFTYPE_1D.

If this field is not set to TRMODE_NONE, the **Surface Format** must be one with 8, 16, 32, 64, or

RENDER_SURFACE_STATE

128 bits per element, or one of the compressed texture modes (BC*, ETC*, EAC*, ASTC*). Additionally, YCRCB* formats are supported and treated as 16 bits per element, and the PLANAR_420_8 format is support and treated as 8 bits per element on the Y plane and 16 bits per element on the UV plane (if **Separate UV Plane Enable** is disabled) or 8 bits per element on the U and V planes (if **Separate UV Plane Enable** is enabled).

If this field is set to TRMODE_NONE, the surface cannot contain any null pages unless **Surface Type** is BUFFER or STRBUF. A BUFFER or STRBUF surface with null pages must have **Surface Base Address** and **Surface Pitch** set to an integer multiple of the element size, and **Surface Format** must be one with 8, 16, 32, 64, or 128 bits per element.

If **Surface Format** is PLANAR, the surface cannot contain any null pages.

17:16 **Reserved**

Format:	MBZ

15 **Reserved**

Format:	MBZ

14 **Coherency Type**

Specifies the type of coherency maintained for this surface.

Value	Name	Description
0h	GPU coherent	Surface memory is kept coherent with GPU threads using GPU read/write ordering rules. Surface memory is backed by system memory but is not kept coherent with CPU (LLC).
1h	IA coherent	Surface memory is kept coherent with CPU (LLC).

Programming Notes

This field may optionally be 1 (IA coherent) for messages sent to SFID_DP_DC0 or SFID_DP_DC1 or SFID_DP_DC2. This field is typically set to 0 (GPU coherent) if the context is operating in a non-SVM legacy mode (for example, Ring Buffer or a Execlist using 32-bit Virtual Address Legacy Context PPGTT32).

If the surface is not IA coherent, then it is possible that data written to Null Tile address will be cached, and then a later read to that same address might return a non-zero value. If a value is cached for a Null Tile address, the data will be eventually be discarded when the cache line is evicted. If the surface is IA coherent, then the cache line never contains a non-zero value, and Null Tile reads always return zero.

13:12 **Reserved**

Format:	MBZ

11:8 **Mip Tail Start LOD**

Format:	U4 in LOD Units

For Sampling Engine, Render Target, and Typed Surfaces:

This field indicates which LOD is the first one in the MIP tail if **Tiled Resource Mode** is not TRMODE_NONE. The MIP tail has a different layout than the rest of the surface. Refer to the

RENDER_SURFACE_STATE

Memory Data Formats section for more details.

For other surfaces:

This field is ignored.

Programming Notes
This field must be zero if the **Surface Format** is MONO8
This field is ignored if **Tiled Resource Mode** is TRMODE_NONE.
If **Tiled Resource Mode** is not TRMODE_NONE, this field must be set to ensure that mips within the mip tail do not overlap given the storage algorithms given in the Memory Data Formats section.
If **Tiled Resource Mode** is not TRMODE_NONE, to disable the Mip Tail this field must be set to a mip that larger than those present in the surface (i.e. 15). This is recommended for non-mip-mapped surfaces.
The following table indicates the *maximum* size of the mip that is set to be the Mip Tail Start LOD for various cases:

Surface Type	Tiling Mode	#MS	Bits Per Element				
			8	16	32	64	128
1D	64KB	1	16384	8192	4096	2048	1024
	4KB	1	1024	512	256	128	64
2D/ CUBE	TIleYS	1	128x256	128x128	64x128	64x64	32x64
		2	128x128	128x64	64x64	64x32	32x32
		4	64x128	64x64	32x64	32x32	16x32
		8	64x64	64x32	32x32	32x16	16x16
		16	32x64	32x32	16x32	16x16	8x16
	TileYF	any	32x64	32x32	16x32	16x16	8x16
3D	TIleYS	1	32x32x32	16x32x32	16x32x16	16x16x16	8x16x16
	TIleYF	1	16x8x16	8x8x16	8x8x8	8x4x8	4x4x8

7:4 **Surface Min LOD**

Format:	U4 In LOD Units

For Sampling Engine and Typed Surfaces:

This field indicates the most detailed LOD that can be accessed as part of this surface. This field is added to the delivered LOD (*sample_l, ld*, or *resinfo* message types) before it is used to address the surface.

For Other Surfaces:

This field is ignored.

Programming Notes
This field must be zero if the **Surface Format** is MONO8

RENDER_SURFACE_STATE

	3:0	**MIP Count / LOD** Format: **Sampling Engine and Typed Surfaces:** U4 in (LOD units - 1) **Render Target Surfaces:** U4 in LOD units Range **Sampling Engine and Typed Surfaces:** [0,14] representing [1,15] MIP levels **Render Target Surfaces:** [0,14] representing LOD **Other Surfaces:** [0] **For Sampling Engine and Typed Surfaces:** This field indicates the number of MIP levels allowed to be accessed starting at **Surface Min LOD**, which must be less than or equal to the number of MIP levels actually stored in memory for this surface. For sample* messages, the mip map access is clamped to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here. For ld* messages, out-of-bounds behavior results for LODs outside of the range specified in this field. **For Render Target Surfaces:** This field defines the MIP level that is currently being rendered into. This is the absolute MIP level on the surface and is not relative to the **Surface Min LOD** field, which is ignored for render target surfaces. **For Other Surfaces:** This field is reserved : MBZ **Programming Notes** The **LOD** of a render target must be the same as the **LOD** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER). For render targets with YUV surface formats, the **LOD** must be zero. For sampling engine surfaces with YCRCB* or PLANAR* surface format, **MIP Count** must be zero.
6	31	**Reserved** Exists If: ([Surface Format] != 'PLANAR') Format: MBZ
	31	**Separate UV Plane Enable** Exists If: ([Surface Format] == 'PLANAR') Format: Enable If enabled, this field indicates that the U and V are present as separate planes. If disabled, the UV data is interleaved on a single plane. **Programming Notes** This field must be disabled (separate UV planes are not supported). See the section "Planar Memory Organization" for a description of how the size and location of the chroma planes (U and V) are calculated.

RENDER_SURFACE_STATE

30 Reserved

Exists If:	([Surface Format] == 'PLANAR')
Format:	MBZ

30:16 Auxiliary Surface QPitch

Exists If:	([Surface Format] != 'PLANAR')
Format:	QPitch[16:2]

This field specifies the distance in rows between array slices on the auxiliary surface.

Value	Name	Description
[4h,1FFFCh]		in multiples of 4 (low 2 bits missing)

Programming Notes
This field must be set to an integer multiple of the **Surface Vertical Alignment**
Software must ensure that this field is set to a value sufficiently large such that the array slices in the auxiliary surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored in memory.
For non-multisampled render target's CCS auxiliary surface, QPitch must be computed with Horizontal Alignment = 128 and Surface Vertical Alignment = 256. These alignments are only for CCS buffer and not for associated render target.

29:16 X Offset for U or UV Plane

Exists If:	([Surface Format] == 'PLANAR')
Format:	U14

This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the U plane or interleaved UV plane, depending on the setting of **Separate UV Plane Enable**.

Programming Notes
This field must be a multiple of 4 (bits 1:0 MBZ).
If **Tiled Resource Mode** is enabled, this field must be a multiple of the tile width in pixels.
Auxiliary Surface Mode is forced to AUX_NONE.

15 Reserved

Format:	MBZ

14 Reserved

Exists If:	([Surface Format] == 'PLANAR')
Format:	MBZ

14:12 Reserved

Exists If:	([Surface Format] != 'PLANAR')
Format:	MBZ

RENDER_SURFACE_STATE

11:3 **Auxiliary Surface Pitch**

Exists If:	([Surface Format] != 'PLANAR')
Format:	U9-1 Pitch in #Tiles

This field specifies the Auxiliary surface pitch in (#Tiles - 1).

Value	Name	Description
[0, 511]		-> [1 tile, 512 tiles]

13:0 **Y Offset for U or UV Plane**

Exists If:	([Surface Format] == 'PLANAR')
Format:	U14

This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the U plane or interleaved UV plane, depending on the setting of **Separate UV Plane Enable**.

Programming Notes
For surfaces where **Surface Format** = PLANAR* and **Separate UV Plane** is Enabled, the Y Offset must be programmed in multiples of **half-rows**. For example, for a surface where Y is physically followed by U and then V in memory, the Y Offset to U plane would be (2*Y-Height). For all other PLANAR YUV formats this is programmed in multiples of full rows.
For all format besides PLANAR_420_* This field must be a multiple of 4 (bits 1:0 MBZ). For formats PLANAR_420_* with separate chroma planes (e.g. YV12) this field must be multiple of 4 if U plane is the first chroma plane after the Y (luma) plane. It can be a multiple of 2 if it is the second chroma plane in memory. For formats PLANAR_420_* with interleaved chroma planes (e.g. NV12) this field can be multiple of 2.
If **Tiled Resource Mode** is enabled, this field must be a multiple of the tile height in rows.
Auxiliary Surface Mode is forced to AUX_NONE.

Workaround
Workaround : For formats PLANAR_420_* when this field is not a multiple of 4 the Out-of-Bounds Supression check must be disabled (HALF_SLICE_CHICKEN3[5] must be set to 1) to avoid false out of bound detection.

2:0 **Auxiliary Surface Mode**

Exists If:	([Surface Format] != 'PLANAR')
Format:	U3

Specifies what type of surface the Auxiliary surface is. The Auxiliary surface has its own base address and pitch, but otherwise shares or overrides other fields set for the primary surface, detailed in the programming notes below.

Value	Name	Description
0h	AUX_NONE	No Auxiliary surface is used
1h	AUX_CCS_D	The Auxiliary surface is a CCS (Color Control Surface) with compression disabled or an MCS with compression enabled, depending on **Number of Multisamples**. MCS (Multisample Control Surface) is a special type of CCS.

RENDER_SURFACE_STATE

2h	AUX_APPEND	The Auxiliary surface is an append buffer
3h	AUX_HIZ	The Auxiliary surface is a hierarchical depth buffer
4h	Reserved	
5h	AUX_CCS_E	The Auxiliary surface is a CCS with compression enabled or an MCS with compression enabled, depending on **Number of Multisamples**.
6h-7h	Reserved	

Programming Notes

The CCS and hierarchical depth Auxiliary surface shares **Height, Width, Depth, Surface Type, Surface Array, Surface Min LOD, MIP Count / LOD, Surface Object Control State, Resource Min LOD**, and **Minimum Array Element** with the primary surface. The hierarchical depth Auxiliary surface uses **Surface Horizontal Alignment** of 16, **Surface Vertical Alignment** of 8, regardless of the primary surface's values for these fields. **X & Y Offset** are set to zero for the purpose of accessing the Auxiliary surface. If this field is set to AUX_HIZ, **Surface Format** must be be one of the following: R32_FLOAT, R24_UNORM_X8_TYPELESS, or R16_UNORM, and the format must match the format used when the surface was used as a depth buffer (with R channel corresponding to D channel).

CCS and hierarchical depth Auxiliary surfaces are TileY with **Tiled Resource Mode** of TRMODE_NONE regardless of the tile mode of the primary surface, and **Mip Tail Start LOD** is ignored for these surfaces.

The CCS Auxiliary surface for non-multisampled render targets has Horizontal Alignment = 128 and Vertical alignment = 64.

The CCS Auxiliary surface for **Number of Multisamples** > 1 uses **Surface Horizontal Alignment** of 16 and **Surface Vertical Alignment** of 4 regardless of the primary surface's values for these fields.

If this field is set to AUX_HIZ, **Number of Multisamples** must be MULTISAMPLECOUNT_1, and Surface Type cannot be SURFTYPE_3D.

If **Number of Multisamples** is MULTISAMPLECOUNT_1, AUX_CCS_E setting is only allowed if **Surface Format** is supported for Render Target Compression. This setting enables render target compression.

If **Number of Multisamples** is MULTISAMPLECOUNT_1, AUX_CCS_D setting is only allowed if **Surface Format** supported for Fast Clear. In addition, if the surface is bound to the sampling engine, **Surface Format** must be supported for Render Target Compression for surfaces bound to the sampling engine. For render target surfaces, this setting disables render target compression. For sampling engine surfaces, this mode behaves the same as AUX_CCS_E.

If **Number of Multisamples** is *not* MULTISAMPLECOUNT_1, both AUX_CCS_E and AUX_CCS_D settings indicate that the auxiliary surface is a multisample control surface (MCS), and multisample compression is enabled.

If **Number of Multisamples** is MULTISAMPLECOUNT_1, and if **Tiled Resource Mode** is NOT TRMODE_NONE, then, if CCS tile is NULL, Render Target Tiles represented by that CCS tile are assumed to be NULL by HW.

RENDER_SURFACE_STATE

7 — 31 — Memory Compression Mode

Distinguishes Vertical from Horizontal compression.

Value	Name	Description
0	Horizontal **[Default]**	
1	Vertical	

30 — Memory Compression Enable

Format:	Enable

This surface may contain compressed or compressible pixels. Memory compression will be attempted for writes to this surface. Reads from this surface will check for compressed data.

Programming Notes
The compression control must have 0 value for non-tileY modes. The Memory Compression Enable can be non-zero only for the surface state that has media messages. That is for 3d case the compression control bits will be 0 in normal surface state but can be non-zero in normal surface state for media messages. E.g. sample_unorm.
The only sampler messages supported with memory compression enabled are *sample_8x8*, *sample_unorm*, and SIMD16 *sample*.
Please refer to vol1a Memory Data Formats chapter >section Media Memory Compression for more details, including format restrictions.

29 — Reserved

Format:	MBZ

28 — Reserved

27:25 — Shader Channel Select Red

Format:	**Shader Channel Select** Enumerated Type

Specifies which surface channel is read or written in the Red shader channel.

Programming Notes
The Shader channel selects also define which shader channels are written to which surface channel. If the Shader channel select is SCS_ZERO or SCS_ONE then it is not written to the surface. If the shader channel select is SCS_RED it is written to the surface red channel and so on. If more than one shader channel select is set to the same surface channel only the first shader channel in RGBA order will be written. Each shader channel select must be set to the same surface channel (R = SCS_RED, G = SCS_GREEN, B = SCS_BLUE, A = SCS_ALPHA) if the surface is accessed via the sampler's sample_unorm* or sample_8x8 messages.
The Shader Channel Select fields do not affect the following sampling engine message types: resinfo, sampleinfo, LOD, and ld_mcs. These messages behave as if each Shader Channel Select is set to the same color surface channel.
For the sampling engine *gather4** messages, the Gather4 Source Channel Select field in the message header defines which channel's Shader Channel Select is used to select the surface channel to be sampled. Other Shader Channel Select fields are ignored.
For the sampling engine *sample*_c* and *gather4*_c* messages, the compare operation always occurs on the red channel from the surface regardless of the setting of the Shader Channel

RENDER_SURFACE_STATE

		Select fields. For Render Target, Red, Green and Blue Shader Channel Selects MUST be such that only valid components can be swapped i.e. only change the order of components in the pixel. Any other values for these Shader Channel Select fields are not valid for Render Targets. This also means that there MUST not be multiple shader channels mapped to the same RT channel. When multiple Channel selects have the same value and shader channel is disabled, disable channel writes 0s to memory. This behavior does not match with Data Port message via HDC. The output channel is undefined if the source is to a channel is not present for the current surface format. For example, If the surface format is R16_float and the shader channel select green specifies green as the source the output is undefined. It should instead select 0 which is the default for a missing color channel..
	24:22	**Shader Channel Select Green** Format: **Shader Channel Select** Enumerated Type See **Shader Channel Select Red** for details.
	21:19	**Shader Channel Select Blue** Format: **Shader Channel Select** Enumerated Type See **Shader Channel Select Red** for details.
	18:16	**Shader Channel Select Alpha** Format: **Shader Channel Select** Enumerated Type See **Shader Channel Select Red** for details. **Programming Notes** For Render Target, this field MUST be programmed to value = SCS_ALPHA.
	15:12	**Reserved** Format: MBZ
	11:0	**Resource Min LOD** Format: U4.8 in LOD units **For Sampling Engine Surfaces:** This field indicates the most detailed LOD that is present in the resource underlying the surface. Refer to the "LOD Computation Pseudocode" section for the use of this field. **For Other Surfaces:** This field is ignored. **Value** / **Name**: [0,14] **Programming Notes** This field must be zero if the **Surface Format** is MONO8 This field must be zero if the **ChromaKey Enable** is enabled in the associated sampler.

RENDER_SURFACE_STATE

8..9	63:0	**Surface Base Address** Format: GraphicsAddress[63:0]SurfaceBase Specifies the byte-aligned base address of the surface. **Programming Notes** • For SURFTYPE_BUFFER render targets, this field specifies the base address of first element of the surface. The surface is interpreted as a simple array of that single element type. The address must be naturally-aligned to the element size (e.g., a buffer containing R32G32B32A32_FLOAT elements must be 16-byte aligned). • For SURFTYPE_BUFFER non-rendertarget surfaces, this field specifies the base address of the first element of the surface, computed in software by adding the surface base address to the byte offset of the element in the buffer. The base address must be aligned to element size. • Linear depth buffer surface base addresses must be 64-byte aligned. Note that while render targets (color) can be SURFTYPE_BUFFER, depth buffers cannot. • Mipmapped surfaces are stored in a "monolithic" (fixed) format, and only require a single address for the base MIP. All other MIPs are positioned relative to the base MIP. • The Base Address for linear (non-tiled) render target surfaces and surfaces accessed with the typed surface read/write data port messages must be element-size aligned for Non-YUV surface formats, or a multiple of 2 element-sizes for YUV surface formats. • Other linear (non-tiled) surfaces have no alignment requirements (byte alignment is sufficient). • For tiled surfaces, the actual start of the surface can be offset from the Surface Base Address by the X Offset and Y Offset fields. Tiles are inherently page-aligned (4K or 64K). • Certain message types used to access surfaces have more stringent alignment requirements. Please refer to the specific data-port message documentation for additional restrictions. Tiled surface base addresses must be 4KB-aligned. Note that only the offsets from Surface Base Address are tiled, Surface Base Address itself is not transformed using the tiling algorithm. Tiled surface base addresses must be tile aligned (64KB aligned for TileYS, 4KB aligned for all other tile modes). For 1D surfaces, the base address must be 64KB aligned if Tiled Resource Mode is TRMODE_64KB, and 4KB aligned if Tiled Resource Mode is TRMODE_4KB. Compressed (BC*, ASTC, etc.) surface data is usually copied by re-describing each MIP/slice as a separate surface, using a size-equivalent RGBA format. But a MIP/slice within a packed **MIP**

RENDER_SURFACE_STATE

		Tail doesn't have the tile-aligned **Surface Base Address** required for the re-description. This case must be specially handled by re-describing the packed **MIP Tail** as a single-MIP surface with the width/pitch/height/depth of a single tile, and then use drawing geometry to "reach out" to the desired tail slot (*x*, *y*, *z*) offset.
10..11	63:62	**Reserved** Exists If: ([Surface Format] == 'PLANAR') Format: MBZ
	61:48	**X Offset for V Plane** Exists If: ([Surface Format] == 'PLANAR') Format: U14 This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the V plane. **Programming Notes** This field must be a multiple of 4 (bits 1:0 MBZ). If **Tiled Resource Mode** is enabled, this field must be a multiple of the tile width in pixels. This field is ignored if **Separate UV Plane Enable** is disabled.
	47:46	**Reserved** Exists If: ([Surface Format] == 'PLANAR') Format: MBZ
	45:32	**Y Offset for V Plane** Exists If: ([Surface Format] == 'PLANAR') Format: U14 This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the V plane. **Programming Notes** For surfaces where **Surface Format** = PLANAR* and **Separate UV Plane** is Enabled, the Y Offset must be programmed in multiples of **half-rows**. For example, for a surface where Y is physically followed by U and then V in memory, the Y Offset to V plane would be (2*Y-Height+ U-Height). For all other PLANAR YUV formats this is programmed in multiples of full rows (e.g Y-Height + U-Height). For all format besides PLANAR_420_* This field must be a multiple of 4 (bits 1:0 MBZ). For formats PLANAR_420_* this field must be multiple of 4 if U plane is the first chroma plane after the Y (luma) plane. It can be a multiple of 2 if it is the second chroma plane. For formats PLANAR_420_* when this field is not a multiple of 4 the Out-of-Bounds Supression check must be disabled (HALF_SLICE_CHICKEN3[5] must be set to 1) to avoid false out of bound detection. If **Tiled Resource Mode** is enabled, this field must be a multiple of the tile height in rows. This field is ignored if **Separate UV Plane Enable** is disabled.

RENDER_SURFACE_STATE

	31:21	**Auxiliary Table Index for Media Compressed Surface** Exists If: [Memory Compression Enable] ==1 This field is valid only if Media Memory Compression is on for the surface(Memory Compression Enable == 1). In that case, the Auxiliary Surface Base address is never expected to be used and hence can be overloaded. This represents the 11 bit index into the table in memory which maps the surface to the auxiliary base address.
	63:12	**Auxiliary Surface Base Address** Exists If: ([Surface Format] != 'PLANAR') AND [Memory Compression Enable] == 0 Format: GraphicsAddress[63:12] Specifies the 4kbyte-aligned base address of the Auxiliary surface associated with the primary surface specified in other SURFACE_STATE fields.
	11	**Reserved** Format: MBZ
	10	**Reserved** Format: MBZ
	9:5	**Quilt Height** Format: U5 This field specifies the height of a quilted texture in units of quilt slices. Refer to the section on Quilted Textures for more details.

Value	Name	Description
[0,31]		representing height of quilt - 1 (y/v dimension)

Programming Notes

Programming Notes

- Only power-of-2 **Quilt Height** and **Quilt Width** values are allowed: (1,2,4,8,16,32) mapping to (0,1,3,7,15,31) values in the fields.
- A surface is defined as a quilted texture if either **Quilt Height** or **Quilt Width** is nonzero (actual field value, not the incremented value).
- A quilted texture
  - is only supported by the sampling engine (other shared functions will ignore the **Quilt Width** and **Quilt Height** field, behaving as if they are set to zero).
  - must have a **Surface Type** of SURFTYPE_2D.
  - must have **Number of Multisamples** set to NUMSAMPLES_1.
  - must have **Vertical Line Stride** set to 0.
  - must have **Auxiliary Surface Mode** set to AUX_NONE.
  - **Depth** indicates the array dimension of the quilted texture if **Surface Array** is enabled. The valid range of **Depth** is [0, 2048 / (QuiltWidth * QuiltHeight) - 1], i.e.

RENDER_SURFACE_STATE

the total number of underlying array slices including quilt slices cannot exceed 2048.

- cannot be accessed with any ld* message type or using a sampler with the **Non-Normalized Coordinate Enable** field enabled.

4:0 Quilt Width

Format:	U5

This field specifies the width of a quilted texture in units of quilt slices. Refer to the section on Quilted Textures for more details.

Value	Name	Description
[0,31]		representing width of quilt - 1 (x/u dimension)

12 31:0 Reserved

Exists If:	([Auxiliary Surface Mode] != 'AUX_CCS_D') AND ([Auxiliary Surface Mode] != 'AUX_CCS_E') AND ([Auxiliary Surface Mode] != 'AUX_HIZ)
Format:	MBZ

31:0 Red Clear Color

Exists If:	[Auxiliary Surface Mode] == 'AUX_CCS_D' OR [Auxiliary Surface Mode] == 'AUX_CCS_E'
Format:	S31 (2's complement) for all SINT surface formats
Format:	U32 for all UINT surface formats
Format:	IEEE_FP for all other surface formats

For Sampling Engine Surfaces and Render Targets with Auxiliary Surface Mode set to AUX_CCS:

Specifies the clear value for the red channel.

For Other Surfaces:

This field is ignored.

Programming Notes
If Number of Multisamples is not MULTISAMPLECOUNT_1, only 0/1 values allowed
If Number of Multisamples is MULTISAMPLECOUNT_1 AND if this RT is fast cleared with non-0/1 clear value, this RT must be partially resolved (refer to Partial Resolve operation) before binding this surface to Sampler.

31:0 Hierarchical Depth Clear Value

Exists If:	[Auxiliary Surface Mode] == 'AUX_HIZ'
Format:	IEEE_Float

If **Auxiliary Surface Mode** is AUX_HIZ, this field specifies the depth clear value associated with this surface. If disabled, this field is ignored.

RENDER_SURFACE_STATE

13	31:0	**Green Clear Color** Exists If: [Auxiliary Surface Mode] == 'AUX_CCS_D' OR [Auxiliary Surface Mode] == 'AUX_CCS_E' Format: S31 (2's complement) for all SINT surface formats Format: U32 for all UINT surface formats Format: IEEE_FP for all other surface formats **For Sampling Engine Surfaces and Render Targets with Auxiliary Surface Mode set to AUX_CCS:** Specifies the clear value for the green channel. **For Other Surfaces:** This field is ignored. **Programming Notes** If programmed to non 0/1 values, SW must ensure a render target partial resolve pass before binding a cleared RT to texture.
	31:0	**Reserved** Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ
14	31:0	**Blue Clear Color** Exists If: [Auxiliary Surface Mode] == 'AUX_CCS_D' OR [Auxiliary Surface Mode] == 'AUX_CCS_E' Format: S31 (2's complement) for all SINT surface formats Format: U32 for all UINT surface formats Format: IEEE_FP for all other surface formats **For Sampling Engine Surfaces and Render Targets with Auxiliary Surface Mode set to AUX_CCS:** Specifies the clear value for the blue channel. **For Other Surfaces:** This field is ignored. **Programming Notes** If programmed to non 0/1 values, SW must ensure a render target partial resolve pass before binding a cleared RT to texture.
	31:0	**Reserved** Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ

RENDER_SURFACE_STATE

15	31:0	**Alpha Clear Color** Exists If: [Auxiliary Surface Mode] == 'AUX_CCS_D' OR [Auxiliary Surface Mode] == 'AUX_CCS_E' Format: S31 (2's complement) for all SINT surface formats Format: U32 for all UINT surface formats Format: IEEE_FP for all other surface formats **For Sampling Engine Surfaces and Render Targets with Auxiliary Surface Mode set to AUX_CCS:** Specifies the clear value for the alpha channel. **For Other Surfaces:** This field is ignored. **Programming Notes** If programmed to non 0/1 values, SW must ensure a render target partial resolve pass before binding a cleared RT to texture.
	31:0	**Reserved** Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ

Render Data Port Message Types

MT_DP_RT - Render Data Port Message Types

Source:	Render Cache DataPort
Size (in bits):	5
Default Value:	0x0000000C

Lists all the Message Types in a Render Data Port Message Descriptor [18:14].

DWord	Bit	Description
0	4	**Reserved** Format: MBZ Ignored
	3:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description
0Ch	MT_RTW **[Default]**	Render Target Write message
0Dh	MT_RTR	Render Target Read message
Others	Reserved	Ignored

Render Target Index Message Header Control

MHC_RT_RTI - Render Target Index Message Header Control

Source:	BSpec
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:3	**Reserved** Format: Ignore Ignored
	2:0	**Render Target Index** Format: U3 Specifies the render target index that will be used to select blend state from BLEND_STATE.

Render Target Message Header

MH_RT - Render Target Message Header

Source: BSpec

Size (in bits): 512

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.0	31:0	**Render Target Controls 0** Format: MHC_RT_C0 Specifies controls for Render Target Write and Read messages.
0.1-0.1	31:0	**Color Calculator State Pointer** Format: MHC_RT_CCSP For Render Target Write message, specifies the HWORD-aligned GeneralStateOffset for Color State. Ignored by Render Target Read message.
0.2-0.2	31:0	**Render Target Index** Format: MHC_RT_RTI For Render Target Write message, specifies the render target index used to select blend state from BLEND_STATE. Ignored by Render Target Read message.
0.3-0.4	63:0	**Reserved** Format: Ignore Ignored
0.5-0.5	31:0	**Color Code** Format: MHC_RT_CC Hardware uses to track synchronizing events and free resources on thread completion.
0.6-0.7	63:0	**Reserved** Format: Ignore Ignored
1.0-1.0	31:0	**Reserved** Format: Ignore Ignored

MH_RT - Render Target Message Header

1.1-1.1	31:0	**Reserved** Format: Ignore Ignored
1.2-1.2	31:0	**Subspan 0** Format: MHC_RT_SUBSPAN Upper left corner of subspan 0
1.3-1.3	31:0	**Subspan 1** Format: MHC_RT_SUBSPAN Upper left corner of subspan 1
1.4-1.4	31:0	**Subspan 2** Format: MHC_RT_SUBSPAN Upper left corner of subspan 2
1.5-1.5	31:0	**Subspan 3** Format: MHC_RT_SUBSPAN Upper left corner of subspan 3
1.6-1.6	31:0	**Reserved** Format: Ignore Ignored
1.7-1.7	31:0	**Pixel Sample Enables** Format: MHC_RT_PSM Pixel Sample Enables

Render Target Message Header Control

MHC_RT_C0 - Render Target Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Reserved** Format: Ignore Ignored
	30:27	**Viewport Index** Format: U4 For Render Target Write message, specifies the index of the viewport currently being used. Range = [0,15] Ignored by Render Target Read message.
	26:16	**Render Target Array Index** Format: U11 Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index. Range = [0,511] SURFTYPE_2D: specifies the array index. Range = [0,511] SURFTYPE_3D: specifies the Z or R coordinate. Range = [0,2047] SURFTYPE_BUFFER: must be zero. SURFTYPE_CUBE: specifies the face identifier. Mapping (0,+x) (1,-x) (2,+y) (3,-y) (4,+z) (5,-z). **Programming Notes** The Render Target Array Index used by hardware for access to the Render Target is overridden with the Minimum Array Element defined in SURFACE_STATE if it is out of the range between Minimum Array Element and Depth. For cube surfaces, a depth value of 5 is used for this determination.
	15	**Front/Back Facing Polygon** Format: U1 Determines whether the polygon is front or back facing. Used by the render cache to determine which stencil test state to use. Value / Name / Description 0h / Front facing / All 1h / Back facing / All
	14	**Stencil Present to Render Target** Format: Enable For Render Target Write message, indicates that computed stencil is included in the message. Must be zero for Render Target Read message.

MHC_RT_C0 - Render Target Message Header Control

	13	**Source Depth Present to Render Target** Format: Enable For Render Target Write Message, indicates that source depth data is included in the message. Must be zero for Render Target Read message.
	12	**oMask to Render Target** Format: Enable For Render Target Write message, indicates that oMask data is present in the message and is to be used to mask off samples. Must be zero for Render Target Read message.
	11	**Source0 Alpha Present to Render Target** Format: Enable For Render Target Write message, indicates that Source0 Alpha (aka o0.a) data is included in RTWrite message. If present, these alpha values are used as inputs to AlphaTest and AlphaToCoverage functions. This is required to meet the API rules when writing to multiple render targets (MRTs). Must be zero for Render Target Read message. **Programming Notes** This bit should not be set when write to RT0, though sending and using redundant alpha will provide the correct results (at lower performance). This bit is not supported on Dual-Source Blend message types, as source0 alpha is already included in those messages. This bit is not supported on replicated data message types.
	10	**Reserved** Format: Ignore Ignored
	9:6	**Starting Sample Pair Index or Sample Index** Format: U4 When pixel shader is dispatched in per-sample mode or per-pixel mode with Per-Sample PS Enable bit cleared, this field indicates the index of the first sample pair of the dispatch. Range = [0,7]. When pixel shader is dispatched in per-pixel mode with Per-Sample PS Enable bit set, this field indicates the index of a sample referenced by per-sample RT read or RT write messages. Range = [0, 15].
	5:0	**Reserved** Format: Ignore Ignored

Replicated Pixel Render Target Data Payload Register

MDPR_RGBA - Replicated Pixel Render Target Data Payload Register

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Red** Format: U32 Specifies the value of all slots' red channel.
1	31:0	**Green** Format: U32 Specifies the value of all slots' green channel.
2	31:0	**Blue** Format: U32 Specifies the value of all slots' blue channel.
3	31:0	**Alpha** Format: U32 Specifies the value of all slots' alpha channel.
4..7	127:0	**Reserved** Format: Ignore Ignored

Replicated SIMD16 Render Target Data Payload

MDP_RTW_16REP - Replicated SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**RGBA** Format: MDPR_RGBA RGBA for all slots [15:0]

Reversed SIMD Mode 2 Message Descriptor Control Field

MDC_SM2R - Reversed SIMD Mode 2 Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
00h	SIMD16	SIMD16
01h	SIMD8	SIMD8

RoundingPrecisionTable_3_Bits

RoundingPrecisionTable_3_Bits

Source: BSpec
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2:0	**Rounding Precision** Format: U3 Value / Name: 000b +1/16 001b +2/16 010b +3/16 011b +4/16 100b +5/16 101b +6/16 110b +7/16 111b +8/16

S0A SIMD8 Render Target Data Payload

MDP_RTW_A8 - S0A SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Red** Format: MDP_DW_SIMD8 Slots [7:0] Red
2.0-2.7	255:0	**Green** Format: MDP_DW_SIMD8 Slots [7:0] Green
3.0-3.7	255:0	**Blue** Format: MDP_DW_SIMD8 Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha

S0A SIMD16 Render Target Data Payload

MDP_RTW_A16 - S0A SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 2560

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_A16 - S0A SIMD16 Render Target Data Payload		
5.0-5.7	255:0	**Green[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Green
6.0-6.7	255:0	**Blue[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Blue
7.0-7.7	255:0	**Blue[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
8.0-8.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
9.0-9.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha

SAMPLER_BORDER_COLOR_STATE

SAMPLER_BORDER_COLOR_STATE

Source:	BSpec
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

Description

The interpretation of the border color depends on the Texture Border Color Mode field in SAMPLER_STATE as follows:

- DX9 mode: The border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used even for the missing channels.
- DX10/OGL mode: the format of the border color depends on the format of the surface being sampled. If the map format is UINT, then the border color format is R32G32B32A32_UINT. If the map format is SINT, then the border color format is R32G32B32A32_SINT. Otherwise, the border color format is R32G32B32A32_FLOAT. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the red channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.The format of this state depends on the Texture Border Color Mode field.

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9

SAMPLER_BORDER_COLOR_STATE

	23:16	**Border Color Blue** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	15:8	**Border Color Green** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	31:0	**Border Color Red - (DX10/0GL)** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/0GL' Format: IEEE_FP Texture Border Color Mode = DX10/OGL
	7:0	**Border Color Red - (DX9)** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
1	31:0	**Border Color Green** Format: IEEE_FP Texture Border Color Mode = DX10/OGL
2	31:0	**Border Color Blue** Format: IEEE_FP Texture Border Color Mode = DX10/OGL
3	31:0	**Border Color Alpha** Format: IEEE_FP Texture Border Color Mode = DX10/OGL

SAMPLER_INDIRECT_STATE_BORDER_COLOR

SAMPLER_INDIRECT_STATE_BORDER_COLOR

Source:	BSpec
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure is a one version of the SAMPLER_INDIRECT_STATE structure, suitable for many needs.
An instance of this structure is pointed to by the **Indirect State Pointer** field in SAMPLER_STATE.
The interpretation of the border color depends on the **Texture Border Color Mode** field in SAMPLER_STATE as follows:

- In **DX9** mode, the border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used *even for the missing channels*.
- In **DX10/OGL** mode, the format of the border color is R32G32B32A32_FLOAT, R32G32B32A32_SINT, or R32G32B32A32_UINT, depending on the surface format chosen. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the *red* channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
	23:16	**Border Color Blue As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8

SAMPLER_INDIRECT_STATE_BORDER_COLOR

	15:8	**Border Color Green As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
	31:0	**Border Color Red As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
	31:0	**Border Color Red As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Red As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	7:0	**Border Color Red As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
1	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Green As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Green As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Green As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
2	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Blue As S31**

SAMPLER_INDIRECT_STATE_BORDER_COLOR

		Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Blue As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Blue As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
3	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Alpha As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Alpha As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Alpha As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float

SAMPLER_INDIRECT_STATE

SAMPLER_INDIRECT_STATE

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Note: There are three variations of this structure, defined separately because their payloads have different lengths. Currently only SAMPLER_INDIRECT_STATE_BORDER_COLOR is fully defined.
This structure is pointed to by **Indirect State Pointer** (SAMPLER_STATE).
The interpretation of the border color depends on the **Texture Border Color Mode** field in SAMPLER_STATE as follows:

- In **DX9** mode, the border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used *even for the missing channels*.
- In **DX10/OGL** mode, the format of the border color is R32G32B32A32_FLOAT, R32G32B32A32_SINT, or R32G32B32A32_UINT, depending on the surface format chosen. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the *red* channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.

The format of this state depends on the **Texture Border Color Mode** field.

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8

SAMPLER_INDIRECT_STATE

		Texture Border Color Mode = DX9
	23:16	**Border Color Blue** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	15:8	**Border Color Green** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	31:0	**Border Color Red** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: SINT32 (2's complement) for all SINT surface formats Format: UINT32 for all UINT surface formats Format: IEEE_FP for all other surface formats Texture Border Color Mode = DX10/OGL
	7:0	**Border Color Red** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
1	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Green** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
2	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ

SAMPLER_INDIRECT_STATE

	31:0	**Border Color Blue** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
3	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Alpha** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
4..15	31:0	**Reserved**

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description

ExistsIf = AVS && (Function_mode = 0)

DWord	Bit	Description
0	31:24	**Table 0Y Filter Coefficient[n,1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Table 0X Filter Coefficient[n,1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Table 0Y Filter Coefficient[n,0]** Format: S1.6 2's Complement **Range:** [-2, +2) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
	7:0	**Table 0X Filter Coefficient[n,0]** Format: S1.6 2's Complement **Range:** [-2, +2) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
1	31:24	**Table 0Y Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 0X Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

	15:8	**Table 0Y Filter Coefficient[n,2]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	7:0	**Table 0X Filter Coefficient[n,2]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
2	31:24	**Table 0Y Filter Coefficient[n,5]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 0X Filter Coefficient[n,5]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	15:8	**Table 0Y Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
	7:0	**Table 0X Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
3	31:24	**Table 0Y Filter Coefficient[n,7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Table 0X Filter Coefficient[n,7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Table 0Y Filter Coefficient[n,6]** Format: S1.6 2's Complement **Range:** [-2, +2)

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

	7:0	**Table 0X Filter Coefficient[n,6]** Format: S1.6 2's Complement **Range:** [-2, +2)
4	31:24	**Table 1X Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 1X Filter Coefficient[n,2]** Format: S1.6 2's Complement **Description** **Range:** [-2.0, +2.0)
	15:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:8	**Table 1X Filter Coefficient[n,5]** Format: S1.6 2's Complement **Description** **Range:** [-2.0, +2.0)
	7:0	**Table 1X Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
6	31:24	**Table 1Y Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 1Y Filter Coefficient[n,2]** Format: S1.6 2's Complement **Description** **Range:** [-2.0, +2.0)
	15:0	**Reserved** Format: MBZ

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

7	31:16	**Reserved** Format: MBZ
	15:8	**Table 1Y Filter Coefficient[n,5]** Format: S1.6 2's Complement **Description** **Range:** [-2.0, +2.0)
	7:0	**Table 1Y Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SAMPLER_STATE_8x8_AVS

SAMPLER_STATE_8x8_AVS	
Source:	BSpec
Size (in bits):	8960
Default Value:	0x0294806C, 0x00000000, 0x39CFD1FF, 0x839F0000, 0x9A6E4000, 0x02601180, 0xFFFE2F2E, 0x00000000, 0xD82E0000, 0x8285ECEC, 0x00008282, 0x00000000, 0x02117000, 0xA38FEC96, 0x00008CC8, 0x00000000, 0x00000000, 0x00000000

SAMPLER_STATE_8x8_AVS

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000

Description
ExistsIf = AVS && (Function_mode = 0)

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:23	**R3c Coefficient** Default Value: 5 Format: U0.5 IEF smoothing coefficient, see IEF map.
	22:18	**R3x Coefficient** Default Value: 5 Format: U0.5 IEF smoothing coefficient, see IEF map.
	17:12	**Strong Edge Threshold** Default Value: 8 Format: U6 If EM > **Strong Edge Threshold**, the basic VSA detects a strong edge.
	11:6	**Weak Edge Threshold** Default Value: 1 Format: U6 If **Strong Edge Threshold** > EM > **Weak Edge Threshold**, the basic VSA detects a weak edge.
	5:0	**Gain Factor** Default Value: 44 Format: U6 User control sharpening strength

SAMPLER_STATE_8x8_AVS

1	31:0	**Reserved** Format: MBZ
2	31:27	**R5c Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, see IEF map.
	26:22	**R5cx Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, see IEF map.
	21:17	**R5x Coefficient** Default Value: 7 Format: U0.5 IEF smoothing coefficient, see IEF map.
	16:14	**Strong Edge Weight** Default Value: 7 Format: U3 Sharpening strength when a strong edge is found in basic VSA.
	13:11	**Regular Weight** Default Value: 2 Format: U3 Sharpening strength when a weak edge is found in basic VSA.
	10:8	**Non Edge Weight** Default Value: 1 Format: U3 Sharpening strength when no edge is found in basic VSA.
	7:0	**Global Noise Estimation** Default Value: 255 Format: U8 Global noise estimation of previous frame.

SAMPLER_STATE_8x8_AVS

3 / 31 — Skin Tone Tuned IEF _ Enable

Default Value:	1
Format:	U1

Control bit to enable the skin tone tuned IEF.

3 / 30 — IEF4Smooth_Enable

Format:	U1

Value	Name	Description
0	**[Default]**	IEF is operating as a content adaptive detail filter based on 5x5 region
1		IEF is operating as a content adaptive smooth filter based on 3x3 region

3 / 29:28 — Enable 8-tap filter

Adaptive Filtering (Mode = 11) ExistsIf:
R10G10B10A2_UNORM R8G8B8A8_UNORM (AYUV also) R8B8G8A8_UNORM B8G8R8A8_UNORM R16G16B16A16

Enable 8-tap Filtering on UV channel (Mode = 10) ExistsIf:
R10G10B10A2_UNORM
R8G8B8A8_UNORM (AYUV also)
R8B8_UNORM (CrCb)
R8_UNORM
R8B8G8A8_UNORM
B8G8R8A8_UNORM
R16G16B16A16
Y8_UNORM

Value	Name	Description
00b		4-tap filter is only done on all channels.
01b		Enable 8-tap Adaptive filter on G-channel. 4-tap filter on other channels.
10b		8-tap filter is done on all channels (UV-ch uses the Y-coefficients)
11b		Enable 8-tap Adaptive filter all channels (UV-ch uses the Y-coefficients).

Programming Notes

For 00 and 10, are applicable for RGB surfaces only or surface without Y-ch.
In case it is a YUV surface it will default to adaptive mode automatically which is 01 and 11 respectively. Alpha channel is always bi-linear filter irrespective of the above modes.

Mode 01 and 00 are legacy support and are supported on all surface formats.

When Mode is 10 and Surface format is Y8_UNORM, Bypass X/Y Adaptive Filtering must be 1, and Default Sharp Level must be 255

SAMPLER_STATE_8x8_AVS

	27:22	**Hue_Max** Default Value: 14 Format: U6 Rectangle half width.
	21:16	**Sat_Max** Default Value: 31 Format: U6 Rectangle half length
	15:8	**Cos(alpha)** Format: S0.7 2's Complement Deafult Value: 79/128
	7:0	**Sin(alpha)** Format: S0.7 2's Complement Deafult Value: 101/128
4	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate.
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate.
	15	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	14:12	**Diamond Margin** Default Value: 4 Format: U3

SAMPLER_STATE_8x8_AVS

11 — Shuffle_OutputWriteback for sample_8x8

Value	Name	Description
0		Writeback same as Original Sample_8x8
1		Writeback of Sample_8x8 Is Modified to Suite Sample_Unorm

10:0 — S3U

Format:	S2.8 2's Complement

Deafult Value: 0/256

5 — 31 — SkinDetailFactor

Format:	S0

This flag bit is in operation only when the control bit **Skin Tone TunedIEF_Enable** is on.

Value	Name	Description
1		sign(SkinDetailFactor) is equal to +1, and the content of the detected skin tone area is not detail revealed.
0		sign(SkinDetailFactor) is equal to -1, and the content of the detected skin tone area is detail revealed.

30:24 — Diamond_du

Default Value:	2
Format:	S6 2's Complement

Rhombus center shift in the sat-direction, relative to the rectangle center.

23:21 — HS_margin

Default Value:	3
Format:	U3

Defines rectangle margin

20:13 — Diamond_alpha

Format:	U2.6

Deafault Value: 100/64
1 / tan(β)

12:7 — Diamond_Th

Default Value:	35
Format:	U6

Half length of the rhombus axis in the sat-direction.

SAMPLER_STATE_8x8_AVS

	6:0	**Diamond_dv** Default Value: 0 Format: S6 2's Complement Rhombus center shift in the hue-direction, relative to the rectangle center.
6	31:24	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function.
	23:16	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	15:8	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	7:0	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
7	31:16	**Reserved** Format: MBZ
	15:0	**INV_Margin_VYL** Format: U0.16 1/Margin_VYL = 3300/65536
8	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.

SAMPLER_STATE_8x8_AVS

	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**INV_Margin_VYU** 1/Margin_VYU = 1600/65536
9	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y Point 2 of the lower part of the detection PWLF.
10	31:27	**Y_Slope_2** Format: U2.3 Deafault Value: 31/8 Slope between points Y3 and Y4.
	26:16	**S0L** Format: S2.8 2's Complement Deafault Value: -5/256 Slope 0 of the lower part of the detection PWLF.

SAMPLER_STATE_8x8_AVS

	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8
11	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Format: S2.8 2's Complement Default Value: 0/256 Slope 2 of the lower part of the detection PWLF.
	10:0	**S1L** Format: S2.8 2's Complement Default Value: 0/256 Slope 1 of the lower part of the detection PWLF.
12	31:27	**Y_Slope1** Format: U2.3 Default Value: 31/8 Slope between points Y1 and Y2.
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.

SAMPLER_STATE_8x8_AVS

	10:0	**S3L** Format: S2.8 2's Complement Default Value: 0/256 Slope 3 of the lower part of the detection PWLF.
13	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
14	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Format: S2.8 2's Complement Default Value: 256/256 Slope 0 of the upper part of the detection PWLF.
	15:8	**B3U** Default Value: 140 Format: U8 V Bias 3 of the upper part of the detection PWLF.

SAMPLER_STATE_8x8_AVS

	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
15	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Format: S2.8 2's Complement Deafult Value: -179/256 Slope 2 of the upper part of the detection PWLF.
	10:0	**S1U** Format: S2.8 2's Complement Deafult Value: 113/256 Slope 1 of the upper part of the detection PWLF.
16..23	255:0	**Filter Coefficient[0]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
24..31	255:0	**Filter Coefficient[1]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
32..39	255:0	**Filter Coefficient[2]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
40..47	255:0	**Filter Coefficient[3]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
48..55	255:0	**Filter Coefficient[4]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
56..63	255:0	**Filter Coefficient[5]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
64..71	255:0	**Filter Coefficient[6]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
72..79	255:0	**Filter Coefficient[7]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
80..87	255:0	**Filter Coefficient[8]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS

SAMPLER_STATE_8x8_AVS

88..95	255:0	**Filter Coefficient[9]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
96..103	255:0	**Filter Coefficient[10]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
104..111	255:0	**Filter Coefficient[11]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
112..119	255:0	**Filter Coefficient[12]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
120..127	255:0	**Filter Coefficient[13]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
128..135	255:0	**Filter Coefficient[14]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
136..143	255:0	**Filter Coefficient[15]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
144..151	255:0	**Filter Coefficient[16]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
152	31:24	**Default Sharpness Level** Format: U8 When adaptive scaling is off, determines the balance between sharp and smooth scalers.

Value	Name	Description
0	**[Default]**	Contribute 1 from the smooth scalar
255		Contribute 1 from the sharp scalar

	23:16	**Max Derivative 4 Pixels** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 4 pixel area.
	15:8	**Max Derivative 8 Pixels** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 8 pixel area.
	7	**Reserved** Format: MBZ
	6:4	**Transition Area with 4 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 4 pixel calculation.

SAMPLER_STATE_8x8_AVS

	3	**Reserved** Format: MBZ
	2:0	**Transition Area with 8 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 8 pixel calculation.
153	31:23	**Reserved** Format: MBZ
	22	**Bypass X Adaptive Filtering** Format: Disable When disabled, the X direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value. **Value** / **Name** / **Description** 1 / Disble / Disable X Adaptive Filtering 0 / Enable / Enable X Adaptive Filtering
	21	**Bypass Y Adaptive Filtering** Format: Disable When disabled, the Y direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value. **Value** / **Name** / **Description** 1 / Disble / Disable Y Adaptive Filtering 0 / Enable / Enable Y Adaptive Filtering
	20:2	**Reserved** Format: MBZ
	1	**Adaptive Filter for all channels** Format: Enable Only to be enabled if 8-tap Adaptive filter mode is on, eElse it should be disabled. **Value** / **Name** / **Description** 1 / Enable / Enable Adaptive Filter on UV/RB Channels 0 / Disble / Disable Adaptive Filter on UV/RB Channels
	0	**RGB Adaptive** Format: Enable This should be always set to 0 for YUV input and can be enabled/disabled for RGB input. This should be enabled only if we enable 8-tap adaptive filter for RGB input. **Value** / **Name** / **Description** 1 / Enable / Enable the RGB Adaptive filter using the equation (Y=(R+2G+B)»2) 0 / Disble / Disable the RGB Adaptive equation and use G-Ch directly for adaptive filter

SAMPLER_STATE_8x8_AVS		
154..159	191:0	**Reserved** Format: MBZ
160..167	255:0	**Filter Coefficient[17]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
168..175	255:0	**Filter Coefficient[18]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
176..183	255:0	**Filter Coefficient[19]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
184..191	255:0	**Filter Coefficient[20]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
192..199	255:0	**Filter Coefficient[21]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
200..207	255:0	**Filter Coefficient[22]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
208..215	255:0	**Filter Coefficient[23]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
216..223	255:0	**Filter Coefficient[24]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
224..231	255:0	**Filter Coefficient[25]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
232..239	255:0	**Filter Coefficient[26]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
240..247	255:0	**Filter Coefficient[27]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
248..255	255:0	**Filter Coefficient[28]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
256..263	255:0	**Filter Coefficient[29]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
264..271	255:0	**Filter Coefficient[30]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS
272..279	255:0	**Filter Coefficient[31]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS

SAMPLER_STATE

SAMPLER_STATE

Source:	BSpec
Exists If:	//(MessageType != 'Deinterlace') && (MessageType != 'Sample_8x8')
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the normal sampler state used by all messages that use SAMPLER_STATE except sample_8x8 and deinterlace. The sampler state is stored as an array of up to 16 elements, each of which contains the dwords described here. The start of each element is spaced 4 dwords apart. The first element of the sampler state array is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31	**Sampler Disable** Format: Disable This field allows the sampler to be disabled. If disabled, all output channels will return 0.
	30	**Reserved**
	29	**Texture Border Color Mode** For some surface formats, the 32 bit border color is decoded differently based on the border color mode. In addition, the default value of channels not included in the surface may be affected by this field. Refer to the "Sampler Output Channel Mapping" table for the values of these channels, and for surface formats that may only support one of these modes. Also refer to the definition of SAMPLER_BORDER_COLOR_STATE for more details on the behavior of the two modes defined by this field.

Value	Name	Description
0h	DX10/OGL	DX10/OGL mode for interpreting the border color
1h	DX9	DX9 and earlier mode for interpreting the border color

Programming Notes
This feild must not be set to DX9 if there are null tiles in use
This field is required to be the same for every message over a period of time. A flush of the sampler cache must occur before a message with the opposite state of this field is delivered.
This field must be set to DX9 mode when used with surfaces that have Surface Format P4A4_UNORM or A4P4_UNORM.
This field must be set to DX10/OGL mode when used with surfaces that have Surface Format YCRCB_SWAPUV or YCRCB_SWAPY.
This field must be set to DX10/OGL mode if **Surface Format** for the associated surface is UINT OR SINT.
This field must be set to DX10/OGL mode if REDUCTION_MINIMUM or REDUCTION_MAXIMUM or message type is sample_min or sample_max.

SAMPLER_STATE

28:27 **LOD PreClamp Mode**

This field determines whether the computed LOD is clamped to [max,min] mip level before the mag-vs-min determination is performed.
PRECLAMP_OGL: LOD pre-clamped to **Min LOD** and **Max LOD**
OpenGL API currently clamps LOD to the **Min LOD** and **Max LOD** (from Sampler State) prior to performing min/mag determination, and therefore it is expected that an OpenGL driver would need to set this field to PRECLAMP_OGL.

Value	Name	Description
0h	NONE	LOD PreClamp disabled
1h	Reserved	
2h	OGL	LOD PreClamp enabled (OGL mode)

26:22 **Coarse LOD Quality Mode**

Format:	U5

This field configures the coarse LOD image quality mode for the sample_d, sample_l, and sample_b messages in the sampling engine. In general, performance will increase and power consumption will decrease with each step of reduced quality (performance gain for sample_l and sample_b will be minimal).

Value	Name	Description
0h	Disabled	Full quality is enabled, matching prior products
01h-1Fh		Quality degrades with each larger value, performance improves with each larger value

Programming Notes
Although allowed, it is not recommended to program this field to a value greater than 17h to avoid masking the exponent which may generate incorrect LOD values.

21:20 **Mip Mode Filter**

Format:	U2 Enumerated Type

This field determines if and how mip map levels are chosen and/or combined when texture filtering.

Value	Name	Description
0h	NONE	Disable mip mapping - force use of the mipmap level corresponding to Min LOD.
1h	NEAREST	Nearest, Select the nearest mip map
2h	Reserved	
3h	LINEAR	Linearly interpolate between nearest mip maps (combined with linear min/mag filters this is analogous to "Trilinear" filtering).

Programming Notes

SAMPLER_STATE

		MIPFILTER_LINEAR is not supported for surface formats that do not support "Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type or minimum/maximum operation.
		Mip Mode Filter must be set to MIPFILTER_NONE or MIPFILTER_NEAREST if Surface Format for the associated surface is UINT or SINT. However, all settings of this field are allowed with UINT/SINT if a minimum or maximum operation is being performed.
		Mip Mode Filter must be set to MIPFILTER_NONE for Planar YUV surfaces.
	19:17	**Mag Mode Filter** Format: U3 Enumerated Type This field determines how texels are sampled/filtered when a texture is being "magnified" (enlarged). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension.

Value	Name	Description
0h	NEAREST	Sample the nearest texel
1h	LINEAR	Bilinearly filter the 4 nearest texels
2h	ANISOTROPIC	Perform an "anisotropic" filter on the chosen mip level
4h-5h	Reserved	
6h	MONO	Perform a monochrome convolution filter
7h	Reserved	

Programming Notes
Only MAPFILTER_NEAREST and MAPFILTER_LINEAR are supported for surfaces of type SURFTYPE_3D.
Only MAPFILTER_NEAREST is supported for surface formats that do not support "Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type or minimum/maximum operation.
MAPFILTER_MONO: Only CLAMP_BORDER texture addressing mode is supported. . Both Mag Mode Filter and Min Mode Filter must be programmed to MAPFILTER_MONO. Mip Mode Filter must be MIPFILTER_NONE. Only valid on surfaces with Surface Format MONO8 and with Surface Type SURFTYPE_2D.
MAPFILTER_ANISOTROPIC may cause artifacts at cube edges if enabled for cube maps with the TEXCOORDMODE_CUBE addressing mode.
MAPFILTER_ANISOTROPIC will be overridden to MAPFILTER_LINEAR when using a sample_l or sample_l_c message type or when Force LOD to Zero is set in the message header.
Both Mag Mode Filter and Min Mode Filter must be set to MAPFILTER_NEAREST if Surface Format for the associated surface is UINT or SINT. However, all settings of this field other than MAPFILTER_MONO are allowed with UINT/SINT if a minimum or maximum operation is being performed.

SAMPLER_STATE

16:14 Min Mode Filter

Format:	U3 Enumerated Type

This field determines how texels are sampled/filtered when a texture is being "minified" (shrunk). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension.See Mag Mode Filter

Value	Name	Description
0h	NEAREST	Sample the nearest texel
1h	LINEAR	Bilinearly filter the 4 nearest texels
2h	ANISOTROPIC	Perform an "anisotropic" filter on the chosen mip level
4h-5h	Reserved	
6h	MONO	Perform a monochrome convolution filter
7h	Reserved	

13:1 Texture LOD Bias

Format:	S4.8 2's complement

Range: [-16.0, 16.0)

This field specifies the signed bias value added to the calculated texture map LOD prior to min-vs-mag determination and mip-level clamping. Assuming mipmapping is enabled, a positive LOD bias will result in a somewhat blurrier image (using less-detailed mip levels) and possibly higher performance, while a negative bias will result in a somewhat crisper image (using more-detailed mip levels) and may lower performance.

Programming Notes

There is no requirement or need to offset the LOD Bias in order to produce a correct LOD for texture filtering (as was required for correct bilinear and anisotropic filtering in some legacy devices).

0 LOD algorithm

Format:	U1 Enumerated Type

Controls which algorithm is used for LOD calculation. Generally, the EWA approximation algorithm results in higher image quality than the legacy algorithm.

Value	Name	Description
0h	LEGACY	Use the legacy algorithm for anisotropic filtering
1h	EWA Approximation	Use the new EWA approximation algorithm for anisotropic filtering

1 31:20 Min LOD

Format:	U4.8 in LOD units

Range: [0.0, 14.0], where the upper limit is also bounded by the Max LOD.

This field specifies the minimum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and

SAMPLER_STATE

before this maximum (resolution) mip clamping is applied.The integer bits of this field are used to control the "maximum" (highest resolution) mipmap level that may be accessed (where LOD 0 is the highest resolution map). The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use.

Programming Notes

If Min LOD is greater than Max LOD, Min LOD takes precedence, i.e. the resulting LOD will always be Min LOD.

This field must be zero if the Min or Mag Mode Filter is set to MAPFILTER_MONO

19:8 **Max LOD**

Format:	U4.8 in LOD units

Range: [0.0, 14.0]

This field specifies the maximum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and before this minimum (resolution) mip clamping is applied.The integer bits of this field are used to control the "minimum" (lowest resolution) mipmap level that may be accessed.The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use.Force the mip map access to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here.

7 **ChromaKey Enable**

Format:	Enable This field enables the chroma key function.

Programming Notes

Supported only on a specific subset of surface formats. See section titled: "Surface Formats" in thsi volume for supported formats.
This field must be disabled if min or mag filter is MAPFILTER_MONO or MAPFILTER_ANISOTROPIC.
This field must be disabled if used with a surface of type SURFTYPE_3D.

6:5 **ChromaKey Index**

Format:	U2

Range: [0, 3]

This field specifies the index of the ChromaKey Table entry associated with this Sampler. This field is a "don't care" unless **ChromaKey Enable** is ENABLED.

4 **ChromaKey Mode**

Format:	U1 Enumerated Type

This field specifies the behavior of the device in the event of a ChromaKey match. This field is ignored if ChromaKey is disabled.

SAMPLER_STATE

KEYFILTER_REPLACE_BLACK:
In this mode, each texel that matches the chroma key is replaced with (0,0,0,0) (black with alpha=0) prior to filtering. For YCrCb surface formats, the black value is A=0, R(Cr)=0x80, G(Y)=0x10, B(Cb)=0x80. This will tend to darken/fade edges of keyed regions. Note that the pixel pipeline must be programmed to use the resulting filtered texel value to gain the intended effect, e.g., handle the case of a totally keyed-out region (filtered texel alpha==0) through use of alpha test, etc.

Value	Name	Description
0h	KEYFILTER_KILL_ON_ANY_MATCH	In this mode, if any contributing texel matches the chroma key, the corresponding pixel mask bit for that pixel is cleared. The result of this operation is observable only if the Killed Pixel Mask Return flag is set on the input message.
1h	KEYFILTER_REPLACE_BLACK	In this mode, each texel that matches the chroma key is replaced with (0,0,0,0) (black with alpha=0) prior to filtering. For YCrCb surface formats, the black value is A=0, R(Cr)=0x80, G(Y)=0x10, B(Cb)=0x80. This will tend to darken/fade edges of keyed regions. Note that the pixel pipeline must be programmed to use the resulting filtered texel value to gain the intended effect, e.g., handle the case of a totally keyed-out region (filtered texel alpha==0) through use of alpha test, etc.

3:1 **Shadow Function**

Format:	U3 Enumerated Type

This field is used for shadow mapping support via the sample_c message type, and specifies the specific comparison operation to be used. The comparison is between the texture sample red channel (except for alpha-only formats which use the alpha channel), and the "ref" value provided in the input message.

Value	Name
0h	PREFILTEROP ALWAYS
1h	PREFILTEROP NEVER
2h	PREFILTEROP LESS
3h	PREFILTEROP EQUAL
4h	PREFILTEROP LEQUAL
5h	PREFILTEROP GREATER
6h	PREFILTEROP NOTEQUAL
7h	PREFILTEROP GEQUAL

SAMPLER_STATE

	0	**Cube Surface Control Mode** Format: U1 Enumerated Type When sampling from a SURFTYPE_CUBE surface, this field controls whether the TC* Address Control Mode fields are interpreted as programmed or overridden to TEXCOORDMODE_CUBE. **Value** / **Name** 0h PROGRAMMED 1h OVERRIDE **Programming Notes** This field must be set to CUBECTRLMODE_PROGRAMMED
2	31:30	**Reserved**
	29:28	**Reserved**
	27:26	**Reserved**
	31:24	**Reserved**
	25:24	**Reserved**
	23:6	**Indirect State Pointer** **Description** This pointer is relative to the Dynamic State Base Address.
	5	**Reserved** Format: MBZ
	4	**Reserved**
	3	**Reserved**
	2	**Reserved**
	1	**Reserved**
	0	**LOD Clamp Magnification Mode** Format: U1 Enumerated Type This field allows the flexibility to control how LOD clamping is handled when in magnification mode. **Value** / **Name** / **Description** 0h MIPNONE When in magnification mode, Sampler will clamp LOD as if the **Mip Mode Filter** is MIPFILTER_NONE. This is how OpenGL defines magnification, and therefore it is expected that those drivers would not set this bit. 1h MIPFILTER When in magnification mode, Sampler will clamp LOD based on the value of **Mip Mode Filter**.
3	31:24	**Reserved**
	25:24	**Reserved** Format: MBZ

SAMPLER_STATE

23:22 **Reduction Type**

Format:	U2 Enumerated Type

This field defines the type of reduction that will be performed on the texels in the footprint defined by the **Min/Mag/Mip Filter Mode** fields. This field is ignored if **Reduction Type Enable** is disabled.

Value	Name	Description
0h	STD_FILTER	standard filter
1h	COMPARISON	comparison followed by standard filter
2h	MINIMUM	minimum of footprint
3h	MAXIMUM	maximum of footprint

Programming Notes
The following message types ignore this field: *sample_min, sample_max, sample_unorm*, resinfo, sampleinfo, LOD, ld*, sample_8x8*.
If the current min/mag filter mode is MAPFILTER_MONO, this field is ignored.
The *sample_c, sample_l_c, sample_d_c, sample_b_c, gather4_c*, and *gather4_po_c* message types, when used with STD_FILTER, MINIMUM, or MAXIMUM settings of this field, perform the operation of the message of the same name without the "_c". The ref parameter is ignored by hardware.
For message types not listed above, when used with COMPARISON setting of this field, perfom the operation of the message of the same name with "_c" included. The ref parameter used by the operation (since it is not delivered in the message) is set to zero.
Restrictions applying to the message whose behavior is being performed must be followed. For example, a sample message used with COMPARISON reduction filter must follow all of the restrictions of *sample_c*. An exception to this is the MINIMUM and MAXIMUM reduction types allow SURFTYPE_1D, 2D, 3D, and CUBE, including with **Surface Array** enabled, even though the sample_min/max messages only allow 2D.
Restrictions applying to the message delivered need not be followed. For example, a *sample_c* message used with STD_FILTER reduction filter needs to follow only the restrictions of sample, not the restrictions of *sample_c*.

21:19 **Maximum Anisotropy**

Format:	U3 Enumerated Type

This field clamps the maximum value of the anisotropy ratio used by the MAPFILTER_ANISOTROPIC filter (Min or Mag Mode Filter).

Value	Name	Description
0h	RATIO 2:1	At most a 2:1 aspect ratio filter is used
1h	RATIO 4:1	At most a 4:1 aspect ratio filter is used
2h	RATIO 6:1	At most a 6:1 aspect ratio filter is used
3h	RATIO 8:1	At most a 8:1 aspect ratio filter is used

SAMPLER_STATE

4h	RATIO 10:1	At most a 10:1 aspect ratio filter is used
5h	RATIO 12:1	At most a 12:1 aspect ratio filter is used
6h	RATIO 14:1	At most a 14:1 aspect ratio filter is used
7h	RATIO 16:1	At most a 16:1 aspect ratio filter is used

18 **U Address Mag Filter Rounding Enable**

Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes

Hardware will **not** force rounding enable.

17 **U Address Min Filter Rounding Enable**

Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes

Hardware will **not** force rounding enable.

16 **V Address Mag Filter Rounding Enable**

Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes

Hardware will **not** force rounding enable.

15 **V Address Min Filter Rounding Enable**

Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes

Hardware will **not** force rounding enable.

14 **R Address Mag Filter Rounding Enable**

Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes

SAMPLER_STATE

	Hardware will **not** force rounding enable.
13	**R Address Min Filter Rounding Enable** Format: Enable Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode. **Programming Notes** Hardware will **not** force rounding enable.
12:11	**Trilinear Filter Quality** Format: U2 Enumerated Type Selects the quality level for the trilinear filter.

Value	Name	Description
0	FULL	Full Quality. Both mip maps are sampled under all circumstances.
1	TRIQUAL_HIGH/MAG_CLAMP_MIPFILTER	High Quality. Same as full quality. When in magnification mode, Sampler will clamp LOD based on the value of **Mip Mode Filter**. If LOD values which are within 12.5% of an integer LOD value are rounded to that value prior to filtering and filtering effectively becomes the same as MIP Nearest.
2	MED	Medium Quality. If the contribution of one mip map is less than 25%, only the other mip map contributes. If LOD values which are within 16.67% of an integer LOD value are rounded to that value prior to filtering and filtering effectively becomes the same as MIP Nearest.
3	LOW	Low Quality. If the contribution of one mip map is less than 37.5%, only the other mip map contributes.p

10	**Non-normalized Coordinate Enable** Format: Enable This field, if enabled, specifies that the input coordinates (U/V/R) are in non-normalized space, where each integer increment is one texel on LOD 0. If disabled, coordinates are normalized, where the range 0 to 1 spans the entire surface. **Programming Notes** The following state must be set as indicated if this field is *enabled*: • TCX/Y/Z Address Control Mode must be TEXCOORDMODE_CLAMP, TEXCOORDMODE_HALF_BORDER, or TEXCOORDMODE_CLAMP_BORDER.

SAMPLER_STATE

- Surface Type must be SURFTYPE_2D or SURFTYPE_3D.
- Mag Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR.
- Min Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR.
- Mip Mode Filter must be MIPFILTER_NONE.
- Min LOD must be 0.
- Max LOD must be 0.
- MIP Count must be 0.
- Surface Min LOD must be 0.
- Texture LOD Bias must be 0.

9 **Reduction Type Enable**

Format:	Enable

This field enables the **Reduction Type** field to modify the behavior of messages based on its setting. If this field is disabled, all messages behave as defined and the **Reduction Type** field is ignored.

8:6 **TCX Address Control Mode**

Format:	**Texture Coordinate Mode** Enumerated Type

Controls how the 1st (TCX, aka U) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). The setting of this field is subject to being overridden by the Cube Surface Control Mode field when sampling from a SURFTYPE_CUBE surface.

Programming Notes
When using cube map texture coordinates, each TC component must have the same Address Control Mode.
When TEXCOORDMODE_CUBE is not used accessing a cube map, the map's Cube Face Enable field must be programmed to 111111b (all faces enabled).
MAPFILTER_MONO: Texture addressing modes must all be set to TEXCOORDMODE_CLAMP_BORDER. The **Border Color** is ignored in this mode, a constant value of 0 is used for border color. Software must pad the border texels within the map itself with 0.
If **Surface Format** is PLANAR*, this field must be set to TEXCOORDMODE_CLAMP.

5:3 **TCY Address Control Mode**

Format:	**Texture Coordinate Mode** Enumerated Type

Controls how the 2nd (TCY, aka V) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). See Address TCX Control Mode above for details

Programming Notes
If this field is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER and a 1D surface is sampled, incorrect blending with the border color in the vertical direction may occur.

SAMPLER_STATE		
	2:0	**TCZ Address Control Mode** Format: **Texture Coordinate Mode** Enumerated Type Controls how the 3rd (TCZ) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror).See Address TCX Control Mode above for details

SCISSOR_RECT

SCISSOR_RECT

Source: RenderCS

Size (in bits): 64

Default Value: 0x00000000, 0x00000000

The viewport-specific state used by the SF unit (SCISSOR_RECT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 2 DWords apart. The location of first element of the array, as specified by Pointer to SCISSOR_RECT, is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31:16	**Scissor Rectangle Y Min** Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates less than Y Min will be clipped out if Scissor Rectangle is enabled. NOTE: If Y Min is set to a value greater than Y Max, all primitives will be discarded for this viewport. Value: [0,16383] — Name:
	15:0	**Scissor Rectangle X Min** Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies X Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) X coordinates less than X Min will be clipped out if Scissor Rectangle is enabled. NOTE: If X Min is set to a value greater than X Max, all primitives will be discarded for this viewport. Value: [0,16383] — Name:
1	31:16	**Scissor Rectangle Y Max** Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than Y Max will be clipped out if Scissor Rectangle is enabled. Value: [0,16383] — Name:
	15:0	**Scissor Rectangle X Max** Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies X Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than X Max will be clipped out if Scissor Rectangle is enabled. Value: 0-16383 — Name:

Scratch Hword Block Message Header

MH_A32_HWB - Scratch Hword Block Message Header

Source: DataPort 0

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..2	95:0	**Reserved** Format: Ignore Ignored
3	31:0	**Per Thread Scratch Space** Format: MHC_PTSS Specifies amount of scratch space used by this thread, for Stateless bounds checking.
4	31:0	**Reserved** Format: Ignore Ignored
5	31:0	**Buffer Base Address** Format: MHC_A32_BBA Specifies the surface address offset page [31:10] for A32 stateless messages.
6..7	63:0	**Reserved** Format: Ignore Ignored

SF_CLIP_VIEWPORT

SF_CLIP_VIEWPORT		
Source:	RenderCS	
Size (in bits):	512	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Viewport Matrix Element m00** Format: IEEE_Float
1	31:0	**Viewport Matrix Element m11** Format: IEEE_Float
2	31:0	**Viewport Matrix Element m22** Format: IEEE_Float
3	31:0	**Viewport Matrix Element m30** Format: IEEE_Float
4	31:0	**Viewport Matrix Element m31** Format: IEEE_Float
5	31:0	**Viewport Matrix Element m32** Format: IEEE_Float
6	31:0	**Reserved** Format: MBZ
7	31:0	**Reserved** Format: MBZ
8	31:0	**X Min Clip Guardband** Format: IEEE_Float . This 32-bit float represents the XMin guardband boundary (normalized to Viewport.XMin == -1.0f). This corresponds to the left boundary of the NDC guardband.
9	31:0	**X Max Clip Guardband** Format: IEEE_Float This 32-bit float represents the XMax guardband boundary (normalized to Viewport..XMax == 1.0f). This corresponds to the right boundary of the NDC guardband.

SF_CLIP_VIEWPORT

10	31:0	**Y Min Clip Guardband** Format: IEEE_Float This 32-bit float represents the YMin guardband boundary (normalized to Viewport.YMin == -1.0f). This corresponds to the bottom boundary of the NDC guardband.
11	31:0	**Y Max Clip Guardband** Format: IEEE_Float This 32-bit float represents the YMax guardband boundary (normalized to Viewport.YMax == 1.0f). This corresponds to the top boundary of the NDC guardband.
12	31:0	**X Min ViewPort** Format: IEEE_Float This 32-bit float represents the Viewport.XMin. This is the X min of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
13	31:0	**X Max ViewPort** Format: IEEE_Float This 32-bit float represents the Viewport.XMax. This is the X max of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
14	31:0	**Y Min ViewPort** Format: IEEE_Float This 32-bit float represents the Viewport.YMin. This is the Y min of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
15	31:0	**Y Max ViewPort** Format: IEEE_Float This 32-bit float represents the Viewport.Ymax. This is the Y max of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.

SF_OUTPUT_ATTRIBUTE_DETAIL

SF_OUTPUT_ATTRIBUTE_DETAIL		
Source:	RenderCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Component Override W** Format: Enable If set, the W component of this output Attribute is overridden by the W component of the constant vector specified by ConstantSource.
	14	**Component Override Z** Format: Enable If set, the Z component of this output Attribute is overridden by the Z component of the constant vector specified by ConstantSource.
	13	**Component Override Y** Format: Enable If set, the Y component of output Attribute is overridden by the Y component of the constant vector specified by ConstantSource.
	12	**Component Override X** Format: Enable If set, the X component of output Attribute is overridden by the X component of the constant vector specified by ConstantSource.
	11	**Swizzle Control Mode** Format: U1 Enumerated Type When Attribute Swizzle Enable is ENABLED, this bit controls whether attributes 0-15 or 16-31 are subject to the following swizzle controls: • Component Override X/Y/Z/W • Constant Source • Swizzle Select • Source Attribute • WrapShortest Enables Note that the Number of SF Output Attributes field specifies how many attributes are output. Note: This field does not impact any functions which provide separate states for all 32 attributes (e.g., Point sprite, Constant interpolation). Note: This field is only valid for the first indexed attribute (Attribute[0]). For all other indices, it is Reserved and MBZ.

SF_OUTPUT_ATTRIBUTE_DETAIL

10:9 Constant Source

Format:	U2 enumerated type

This state selects a constant vector which can be used to override individual components of this Attribute

Value	Name	Description
0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0
1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0
2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0
3h	PRIM_ID	Constant.xyzw = PrimID (replicated)

8 Reserved

Format:	MBZ

7:6 Swizzle Select

Format:	U2 enumerated type

This state, along with Source Attribute, specifies the source for this output Attribute.

Value	Name	Description
0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]
1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].
2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute]. The W component is copied to the X component.
3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.

5 Reserved

Format:	MBZ

4:0 Source Attribute

Format:	U5

This field selects the source attribute for this Attribute. Source attribute 0 corresponds to the first 128 bits of data indicated by Vertex URB Entry Read Offset

SFC_8x8_AVS_COEFFICIENTS

SFC_8x8_AVS_COEFFICIENTS

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description

ExistsIf = AVS && (Function_mode = 0)

DWord	Bit	Description
0	31:24	**ZeroYFilterCoefficient1** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient1** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient0** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient0** Format: S1.6 2's Complement **Range:** [-2, +2)
1	31:24	**ZeroYFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-2, +2)

SFC_8x8_AVS_COEFFICIENTS		
	7:0	**ZeroXFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-2, +2)
2	31:24	**ZeroYFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2, +2)
3	31:24	**ZeroYFilterCoefficient7** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient7** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient6** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient6** Format: S1.6 2's Complement **Range:** [-2, +2)
4	31:24	**OneXFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SFC_8x8_AVS_COEFFICIENTS		
	23:16	**OneXFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	15:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:8	**OneXFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	7:0	**OneXFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
6	31:24	**OneYFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**OneYFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	15:0	**Reserved** Format: MBZ
7	31:16	**Reserved** Format: MBZ
	15:8	**OneYFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	7:0	**OneYFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SIMD4x2 MSAA Typed Surface 32-Bit Address Payload

MAP32B_MSAA_TS_SIMD4X2 - SIMD4x2 MSAA Typed Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.0	31:0	**Slot0 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 0
0.1-0.3	95:0	**Reserved** Format: Ignore Ignored
0.4-0.4	31:0	**Slot1 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 1
0.5-0.7	95:0	**Reserved** Format: MBZ Ignored
1.0-1.0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1.1-1.1	31:0	**V0** Format: U32 Specifies the V channel address offset for slot 0.
1.2-1.2	31:0	**R0** Format: U32 Specifies the R channel address offset for slot 0.

MAP32B_MSAA_TS_SIMD4X2 - SIMD4x2 MSAA Typed Surface 32-Bit Address Payload

1.3-1.3	31:0	**LOD0** Format: MACD_LOD Specifies the LOD for slot 0.
1.4-1.4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
1.5-1.5	31:0	**V1** Format: U32 Specifies the V channel address offset for slot 1.
1.6-1.6	31:0	**R1** Format: U32 Specifies the R channel address offset for slot 1.
1.7-1.7	31:0	**LOD1** Format: MACD_LOD Specifies the LOD for slot 1.

SIMD4x2 Typed Surface 32-Bit Address Payload

MAP32B_TS_SIMD4X2 - SIMD4x2 Typed Surface 32-Bit Address Payload

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1	31:0	**V0** Format: U32 Specifies the V channel address offset for slot 0.
2	31:0	**R0** Format: U32 Specifies the R channel address offset for slot 0.
3	31:0	**LOD0** Format: MACD_LOD Specifies the LOD for slot 0.
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5	31:0	**V1** Format: U32 Specifies the V channel address offset for slot 1.
6	31:0	**R1** Format: U32 Specifies the R channel address offset for slot 1.
7	31:0	**LOD1** Format: MACD_LOD Specifies the LOD for slot 1.

SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD4X2 - SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1..3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5..7	95:0	**Reserved** Format: Ignore Ignored

SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD4X2 - SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**U0** Format: U64 Specifies the U channel address offset for slot 0.
2..3	63:0	**Reserved** Format: Ignore Ignored
4..5	63:0	**U1** Format: U64 Specifies the U channel address offset for slot 1.
6..7	63:0	**Reserved** Format: Ignore Ignored

SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD4X2 - SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1	31:0	**V0** Format: U32 Specifies the V channel address offset for slot 0.
2..3	63:0	**Reserved** Format: Ignore Ignored
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5	31:0	**V1** Format: U32 Specifies the V channel address offset for slot 1.
6..7	63:0	**Reserved** Format: Ignore Ignored

SIMD4x2 32-Bit Address Payload

MAP32B_SIMD4X2 - SIMD4x2 32-Bit Address Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Offset0** Format: U32 Specifies the address offset for slot 0.
1..3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**Offset1** Format: U32 Specifies the address offset for slot 1.
5..7	95:0	**Reserved** Format: Ignore Ignored

SIMD8 Dual Source Render Target Data Payload

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload

Source: BSpec

Size (in bits): 2048

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload		
6.0-6.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha

SIMD8 LOD Message Address Payload Control

MACR_LOD_SIMD8 - SIMD8 LOD Message Address Payload Control

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Slot0 LOD** Format: MACD_LOD Specifies the LOD for slot 0
0.1	31:0	**Slot1 LOD** Format: MACD_LOD Specifies the LOD for slot 1
0.2	31:0	**Slot2 LOD** Format: MACD_LOD Specifies the LOD for slot 2
0.3	31:0	**Slot3 LOD** Format: MACD_LOD Specifies the LOD for slot 3
0.4	31:0	**Slot4 LOD** Format: MACD_LOD Specifies the LOD for slot 4
0.5	31:0	**Slot5 LOD** Format: MACD_LOD Specifies the LOD for slot 5
0.6	31:0	**Slot6 LOD** Format: MACD_LOD Specifies the LOD for slot 6
0.7	31:0	**Slot7 LOD** Format: MACD_LOD Specifies the LOD for slot 7

SIMD8 MSAA Typed Surface 32-Bit Address Payload

MAP32B_MSAA_TS_SIMD8 - SIMD8 MSAA Typed Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Slot0 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 0
0.1	31:0	**Slot1 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 1
0.2	31:0	**Slot2 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 2
0.3	31:0	**Slot3 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 3
0.4	31:0	**Slot4 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 4
0.5	31:0	**Slot5 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 5

MAP32B_MSAA_TS_SIMD8 - SIMD8 MSAA Typed Surface 32-Bit Address Payload

0.6	31:0	**Slot6 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 6
0.7	31:0	**Slot7 Sample Number** Format: MACD_MSAA_SN Specifies the sample number for slot 7
1.0-1.7	255:0	**U** Format: MACR_32b Specifies the U channel for slots [7:0]
2.0-2.7	255:0	**V** Format: MACR_32b Specifies the V channel for slots [7:0]
3.0-3.7	255:0	**R** Format: MACR_32b Specifies the R channel for slots [7:0]
4.0-4.7	255:0	**LOD** Format: MACR_LOD_SIMD8 Specifies the LOD for slots [7:0]

SIMD8 Render Target Data Payload

MDP_RTW_8 - SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDP_DW_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDP_DW_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDP_DW_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha

SIMD8 Typed Surface 32-Bit Address Payload

MAP32B_TS_SIMD8 - SIMD8 Typed Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Format: **MACR_32b** Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**V** Format: **MACR_32b** Specifies the V channel for slots [7:0]
2.0-2.7	255:0	**R** Format: **MACR_32b** Specifies the R channel for slots [7:0]
3.0-3.7	255:0	**LOD** Format: **MACR_LOD_SIMD8** Specifies the LOD for slots [7:0]

SIMD8 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Format: MACR_32b Specifies the U channel for slots [7:0]

SIMD8 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 64-Bit Address Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U3_U0** Format: **MACR_64b** Specifies the U channel for slots [3:0]
1.0-1.7	255:0	**U7_U4** Format: **MACR_64b** Specifies the U channel for slots [7:4]

SIMD8 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD8 - SIMD8 Untyped STRBUF Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Format: **MACR_32b** Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**V** Format: **MACR_32b** Specifies the V channel for slots [7:0]

SIMD16 Render Target Data Payload

MDP_RTW_16 - SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 2048

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_16 - SIMD16 Render Target Data Payload		
6.0-6.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
7.0-7.7	255:0	**Alpha[15:7]** Format: MDP_DW_SIMD8 Slots [15:7] Alpha

SIMD16 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U[7:0]** Format: **MACR_32b** Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**U[15:8]** Format: **MACR_32b** Specifies the U channel for slots [15:8]

SIMD16 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 64-Bit Address Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U3_U0** Format: **MACR_64b** Specifies the U channel for slots [3:0]
1.0-1.7	255:0	**U7_U4** Format: **MACR_64b** Specifies the U channel for slots [7:4]
2.0-2.7	255:0	**U11_U8** Format: **MACR_64b** Specifies the U channel for slots [11:8]
3.0-3.7	255:0	**U15_U12** Format: **MACR_64b** Specifies the U channel for slots [15:12]

SIMD16 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD16 - SIMD16 Untyped STRBUF Surface 32-Bit Address Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U7_U0** Format: MACR_32b Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**U15_U8** Format: MACR_32b Specifies the U channel for slots [15:8]
2.0-2.7	255:0	**V7_V0** Format: MACR_32b Specifies the V channel for slots [7:0]
3.0-3.7	255:0	**V15_V8** Format: MACR_32b Specifies the V channel for slots [15:8]

SIMD 32-Bit Address Payload Control

MACR_32B - SIMD 32-Bit Address Payload Control

Source: BSpec
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Offset0** Format: U32 Specifies the address offset for slot 0 in this payload register.
0.1	31:0	**Offset1** Format: U32 Specifies the address offset for slot 1 in this payload register.
0.2	31:0	**Offset2** Format: U32 Specifies the address offset for slot 2 in this payload register.
0.3	31:0	**Offset3** Format: U32 Specifies the address offset for slot 3 in this payload register.
0.4	31:0	**Offset4** Format: U32 Specifies the address offset for slot 4 in this payload register.
0.5	31:0	**Offset5** Format: U32 Specifies the address offset for slot 5 in this payload register.
0.6	31:0	**Offset6** Format: U32 Specifies the address offset for slot 6 in this payload register.
0.7	31:0	**Offset7** Format: U32 Specifies the address offset for slot 7 in this payload register.

SIMD 64-Bit Address Payload Control

MACR_64B - SIMD 64-Bit Address Payload Control

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Offset0** Format: U64 Specifies the address offset for slot 0 in this payload register.
0.2-0.3	63:0	**Offset1** Format: U64 Specifies the address offset for slot 1 in this payload register.
0.4-0.5	63:0	**Offset2** Format: U64 Specifies the address offset for slot 2 in this payload register.
0.6-0.7	63:0	**Offset3** Format: U64 Specifies the address offset for slot 3 in this payload register.

SIMD8 32-Bit Address Payload

MAP32B_SIMD8 - SIMD8 32-Bit Address Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Offset[7:0]** Format: **MACR_32b** Specifies the address offset for Slots [7:0].

SIMD8 64-Bit Address Payload

MAP64B_SIMD8 - SIMD8 64-Bit Address Payload

Source:	BSpec
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[3:0]** Format: **MACR_64b** Specifies the address offset for slots [3:0].
1.0-1.7	255:0	**Offset[7:4]** Format: **MACR_64b** Specifies the address offset for slots [7:4].

SIMD16 32-Bit Address Payload

MAP32B_SIMD16 - SIMD16 32-Bit Address Payload		
Source:	BSpec	
Size (in bits):	512	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Offset[7:0]** Format: **MACR_32b** Specifies the address offset for slots [7:0].
1.0-1.7	255:0	**Offset[15:8]** Format: **MACR_32b** Specifies the address offset for slots [15:8].

SIMD16 64-Bit Address Payload

MAP64B_SIMD16 - SIMD16 64-Bit Address Payload

Source:	BSpec
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[3:0]** Format: **MACR_64b** Specifies the address offsets for slots [3:0].
1.0-1.7	255:0	**Offset[7:4]** Format: **MACR_64b** Specifies the address offsets for slots [7:4].
2.0-2.7	255:0	**Offset[11:8]** Format: **MACR_64b** Specifies the address offsets for slots [11:8].
3.0-3.7	255:0	**Offset[15:12]** Format: **MACR_64b** Specifies the address offsets for slots [15:12].

SIMD Mode 2 Message Descriptor Control Field

MDC_SM2 - SIMD Mode 2 Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
00h	SIMD8	SIMD8
01h	SIMD16	SIMD16

SIMD Mode 3 Message Descriptor Control Field

MDC_SM3 - SIMD Mode 3 Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
00h	SIMD4x2	SIMD4x2
01h	SIMD16	SIMD16
02h	SIMD8	SIMD8
03h	Reserved	Ignored

SLM Surface Pixel Mask Message Header

MH1_SLM_PSM - SLM Surface Pixel Mask Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0..6	223:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

Slot Group 2 Message Descriptor Control Field

MDC_SG2 - Slot Group 2 Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Format: Enumeration Controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present. Value / Name / Description 00h / SG8L / Use low 8 slots 01h / SG8U / Use high 8 slots

Slot Group 3 Message Descriptor Control Field

MDC_SG3 - Slot Group 3 Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**SIMD Mode** Format: Enumeration Controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present.

Value	Name	Description
00h	SG4x2	SIMD4x2
01h	SG8L	Use low 8 slots
02h	SG8U	Use high 8 slots
03h	Reserved	Ignored

Slot Group Select Render Cache Message Descriptor Control Field

MDC_RT_SGS - Slot Group Select Render Cache Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Slot Group Select** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the antialias alpha, multisample coverage mask, and if the header is not present also includes the X/Y addresses and pixel enables. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed.

Value	Name	Description
00h	SLOTGRP_LO	Choose bypassed data for slots 15:0
01h	SLOTGRP_HI	Choose bypassed data for slots 31:16

SO_DECL

SO_DECL

Source: RenderCS

Size (in bits): 16

Default Value: 0x00000000

A list of SO_DECL structures are passed in the 3DSTATE_SO_DECL_LIST command. Each structure specifies either (a) the source and destination of an up-to-4-DWord appending write into an SO buffer, or (b) how many DWords to skip over in the destination SO buffer (i.e., a "hole" where the previous buffer contents are maintained).

DWord	Bit	Description
0	15:14	**Reserved** Format: MBZ
	13:12	**Output Buffer Slot** Format: U2 Buffer Index This field selects the destination output buffer slot.
	11	**Hole Flag** Format: Flag If set, the Component Mask field indirectly specifies a number of 32-bit locations to skip over (leave unmodified in memory) in the selected output buffer. The Register Index field is ignored. The only permitted Component Mask values are as follows: 0x0 No Dwords are skipped over (SO_DECL performs no operation) 0x1 (X) Skip 1 DWord 0x3 (XY) Skip 2 DWords 0x7 (XYZ) Skip 3 DWords 0xF (XYZW) Skip 4 DWords
	10	**Reserved** Format: MBZ
	9:4	**Register Index** Format: U6 128-bit granular offset into the source vertex read data If Hole Flag is clear, this field specifies the 128-bit offset into the source vertex data which supplies the source data to be written to the destination buffer, where the individual 32-component destination locations are selected by Component Mask. e.g., Register Index 0 corresponds with the first 128 bits of the data read from the vertex URB entry (as per corresponding Vertex Read Offset state) There is only enough internal storage for the 128-bit vertex header and 32 128-bit vertex attributes.

SO_DECL

Value	Name
[0,32]	
0h	**[Default]**

Programming Notes
It is the responsibility of software to map any API-visible source data specifications (e.g., vertex register number) into 128-bit granular URB read offsets.

3:0 **Component Mask**

Format:	MASK 4-bit Mask

This field is a 4-bit bitmask that selects which contiguous 32-bit component(s) are either written or skipped-over in the destination buffer.

If this field is zero the SO_DECL operation is effectively a no-op. No data will be appended to the destination and the destination buffer's write pointer will not be advanced.

If the **Hole Flag** is set, this field (if non-zero) indirectly specifies how much the destination buffer's write pointer should be advanced.See **Hole Flag** description above for restrictions on this field.

If the **Hole Flag** is clear, this field (if non-zero) selects which source components are to be written to the destination buffer. The components must be contiguous, e.g. YZW is legal, but XZW is not. The selected source components are written to the destination buffer starting at the current write pointer, and then the write pointer is advanced past the written data. E.g., if YZW is specified, the three (YZW) components of the source register will be written to the destination buffer at the current write pointer, and the write pointer will be advanced by 3 DWords.

Value	Name
0h	**[Default]**
xxx1b	SO_DECL_COMPMASK_X
xx1xb	SO_DECL_COMPMASK_Y
x1xxb	SO_DECL_COMPMASK_Z
1xxxb	SO_DECL_COMPMASK_W

SO_DECL_ENTRY

SO_DECL_ENTRY		
Source:	RenderCS	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:48	**Stream 3 Decl** Format: SO_DECL This field contains Stream 3 SO_DECL [n]
	47:32	**Stream 2 Decl** Format: SO_DECL This field contains Stream 2 SO_DECL [n]
	31:16	**Stream 1 Decl** Format: SO_DECL This field contains Stream 1 SO_DECL [n]
	15:0	**Stream 0 Decl** Format: SO_DECL This field contains Stream 0 SO_DECL [n]

Split_coding_unit_flags

Split_coding_unit_flags		
Source:	VideoCS	
Size (in bits):	21	
Default Value:	0x00000000	
Contains the split level flags, level 0 through 2.		
DWord	**Bit**	**Description**
0	20	**Split_flag_level0** Format: U1
	19:16	**Split_flag_level1** Format: U4 [19:16] is in raster order. Bit16 is for partition0 in raster order.
	15:12	**Split_flag_level2 level1part3** Format: U4 Split flags for bit19 partition. [15:12] is in raster order. Bit12 is for partition0 in raster order.
	11:8	**Split_flag_level2 level1part2** Format: U4 Split flags for bit18 partition. [11:8] is in raster order. Bit8 is for partition0 in raster order.
	7:4	**Split_flag_level2 level1part1** Format: U4 Split flags for bit17 partition. [7:4] is in raster order. Bit4 is for partition0 in raster order.
	3:0	**Split_flag_level2 level1part0** Format: U4 Split flags for bit16 partition. [3:0] is in raster order. Bit0 is for partition0 in raster order.

SplitBaseAddress4KByteAligned

SplitBaseAddress4KByteAligned		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Specifies a 64-bit (48-bit canonical) 4K-byte aligned memory base address.		
DWord	**Bit**	**Description**
0	31:12	**Base Address Low** Format: GraphicsAddress[31:12]
	11:0	**Reserved** Format: MBZ
1 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example, GraphicsAddress[47:12] is a 4KB page address.	31:16	**Reserved** Format: MBZ
	15:0	**Base Address High** Format: GraphicsAddress[47:32]

SplitBaseAddress64ByteAligned

SplitBaseAddress64ByteAligned		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Specifies a 64-bit (48-bit canonical) 64-byte aligned memory base address.		
DWord	**Bit**	**Description**
0..1	63:6	**Base Address** Format: GraphicsAddress63-6
	5:0	**Base Address High** Format: MBZ

SrcRegNum

SrcRegNum

Source: Eulsa

Size (in bits): 8

Default Value: 0x00000000

Description

Register Number
The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number.
An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers.
RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero.
This field is present for the direct addressing mode and not present for indirect addressing.
This field applies to both source and destination operands.

DWord	Bit	Description
0	7:0	**Source Register Number**

Value	Name	Description
0-127	If {Dst/Src0/Src1/Src2}.RegFile==GRF	
0-0ffh	If {Dst/Src0/Src1/Src2}.RegFile==ARF	This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

SrcSubRegNum

SrcSubRegNum

Source:	Eulsa
Size (in bits):	5
Default Value:	0x00000000

Description

Subregister Number
The subregister number for the operand. For a GRF register, is the byte address within a 256-bit (32-byte) register. For an ARF register, determines the sub-register number according to the specified encoding for the given architecture register.
RegNum and SubRegNum together provide the byte-aligned address for the origin of a GRF register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero.

Programming Notes

Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

DWord	Bit	Description
0	4:0	**Source Sub Register Number**

Value	Name	Description
0-31	If {Dst/Src0/Src1/Src2}.RegFile==GRF	
0-0ffh	If {Dst/Src0/Src1/Src2}.RegFile==ARF	This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

SRD Interrupt Bit Definition

SRD Interrupt Bit Definition		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
The SRD Interrupt Registers all share the same bit definitions from this table.		
DWord	**Bit**	**Description**
0	31:30	**Reserved**
	29:27	**Reserved**
	26	**Reserved**
	25	**SRD_Exit_C** This event occurs on the first blank start after SRD exit on transcoder C.
	24	**SRD_PreWarn_C** This event occurs two display frames prior to entering SRD on transcoder C.
	23	**Push Done C** This event occurs after double buffer update or Push done. After this interrupt, Logic is ready to receive another push frame indication.
	23	**Reserved**
	22:20	**Reserved**
	19	**Push Done B** This event occurs after double buffer update or Push done. After this interrupt, Logic is ready to receive another push frame indication.
	19	**Reserved**
	18	**Reserved**
	17	**SRD_Exit_B** This event occurs on the first blank start after SRD exit on transcoder B.
	16	**SRD_PreWarn_B** This event occurs two display frames prior to entering SRD on transcoder B.
	15:12	**Reserved**
	11	**Push Done A** This event occurs after double buffer update or Push done. After this interrupt, Logic is ready to receive another push frame indication.
	11	**Reserved**
	10	**Reserved**
	9	**SRD_Exit_A** This event occurs on the first blank start after SRD exit on transcoder A.
	8	**SRD_PreWarn_A** This event occurs two display frames prior to entering SRD on transcoder A.

SRD Interrupt Bit Definition		
	7:4	**Reserved**
	3	**Push Done EDP** This event occurs after double buffer update or Push done. After this interrupt, Logic is ready to receive another push frame indication.
	3	**Reserved**
	2	**SRD_Aux_Error_EDP** This event occurs on the rising edge of the SRD Aux error (receive error or timeout) indication.
	1	**SRD_Exit_EDP** This event occurs on the first blank start after SRD exit on transcoder EDP.
	0	**SRD_PreWarn_EDP** This event occurs two display frames prior to entering SRD on transcoder EDP.

Stateless Binding Table Index Message Descriptor Control Field

MDC_STATELESS - Stateless Binding Table Index Message Descriptor Control Field

Source: BSpec
Size (in bits): 8
Default Value: 0x000000FF

DWord	Bit	Description
0	7:0	**Binding Table Index** Format: Enumeration Specifies the message is Stateless

Value	Name	Description
0FFh	A32_A64 **[Default]**	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).
Others	Reserved	Ignored

Restriction
Restriction : When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Stateless Block Message Header

MH_A32_GO - Stateless Block Message Header

Source: DataPort 0

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:0	**Reserved** Format: MBZ Ignored
2	31:0	**Global Offset** Format: U32 Specifies the global element index into the buffer, in units of Owords, Dwords, or Bytes (depending on the message). **Programming Notes** If the address offset calculated with the Buffer Base Address and Global Offset is greater than the PTSS size or the GeneralStateBufferSize, then the access is Out-of-Bounds.
3	31:0	**Per Thread Scratch Space** Format: MHC_PTSS Specifies amount of scratch space used by this thread, for Stateless bounds checking.
4	31:0	**Reserved** Format: MBZ Ignored
5	31:0	**Buffer Base Address** Format: MHC_A32_BBA **Description** Specifies the surface address offset page [31:10] for A32 stateless messages. Restriction : When using stateless A32 Data Port messages, General State Base Address[47:12] + Buffer Base Address[31:10] must be less than 2^48. It is illegal for this to be greater or equal than 2^48.
6..7	63:0	**Reserved** Format: MBZ Ignored

Stateless Scaled Data Port 2 Message Header

MH2_A32 - Stateless Scaled Data Port 2 Message Header

Source: DataPort 2

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Surface Pitch** Format: MHC_PITCH Specifies the 16-bit surface pitch to use with scaled A32 stateless messages.
1..4	127:0	**Reserved** Format: Ignore Ignored
5	31:2	**Buffer Base Address** Format: GeneralStateOffset[31:2] Specifies the surface address offset page [31:0] for scaled A32 stateless messages. The address must be Dword aligned (bits [1:0] = 0).
	1:0	**Reserved** Format: MBZ Ignored
6..7	63:0	**Reserved** Format: Ignore Ignored

Stateless Scaled Data Port 2 Pixel Mask Message Header

MH2_A32_PSM - Stateless Scaled Data Port 2 Pixel Mask Message Header

Source:	DataPort 2
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0	31:0	**Surface Pitch** Format: MHC_PITCH Specifies the 16-bit surface pitch to use with scaled A32 stateless messages.
1..4	127:0	**Reserved** Format: Ignore Ignored
5	31:0	**Buffer Base Address** Format: GeneralStateOffset[31:0] Specifies the surface address offset page [31:0] for scaled A32 stateless messages. The address must be Dword aligned (bits [1:0] = 0).
6	31:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 untyped surfaces.

Stateless Surface Message Header

MH1_A32 - Stateless Surface Message Header		
Source:	DataPort 1	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..4	159:0	**Reserved** Format: Ignore Ignored
5	31:0	**Buffer Base Address** Format: MHC_A32_BBA Specifies the surface address offset page [31:10] for A32 stateless messages.
6..7	63:0	**Reserved** Format: Ignore Ignored

Stateless Surface Pixel Mask Message Header

MH1_A32_PSM - Stateless Surface Pixel Mask Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0..4	159:0	**Reserved** Format: Ignore Ignored
5	31:0	**Buffer Base Address** Format: MHC_A32_BBA Specifies the surface address offset page [31:10] for A32 stateless messages.
6	31:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

Stencil Message Data Payload Register

MDPR_STENCIL - Stencil Message Data Payload Register

Source: BSpec

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:24	**Stencil3** Format: U8 Stencil for Slot 3.
	23:16	**Stencil2** Format: U8 Stencil for Slot 2.
	15:8	**Stencil1** Format: U8 Stencil for Slot 1.
	7:0	**Stencil0** Format: U8 Stencil for Slot 0.
1	31:24	**Stencil7** Format: U8 Stencil for Slot 7.
	23:16	**Stencil6** Format: U8 Stencil for Slot 6.
	15:8	**Stencil5** Format: U8 Stencil for Slot 5.
	7:0	**Stencil4** Format: U8 Stencil for Slot 4.

MDPR_STENCIL - Stencil Message Data Payload Register		
2..7	191:0	**Reserved** Format: Ignore Ignored

Subset Atomic Integer Trinary Operation Message Descriptor Control Field

MDC_AOP3S - Subset Atomic Integer Trinary Operation Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x0000000E

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Format: Enumeration Specifies the atomic integer trinary operation to be performed

Value	Name	Description
0Eh	AOP_CMPWR **[Default]**	new_dst = (src0 == old_dst) ? src1 : old_dst
Others	Reserved	Ignored

Programming Notes
When Return Data Control is set, old_dst is returned.

Subset Reversed SIMD Mode 2 Message Descriptor Control Field

MDC_SM2RS - Subset Reversed SIMD Mode 2 Message Descriptor Control Field

Source:	BSpec
Size (in bits):	1
Default Value:	0x00000001

DWord	Bit	Description
0	0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
0h	Reserved	Not used
01h	SIMD8 **[Default]**	SIMD8

Subset SIMD Mode 2 Message Descriptor Control Field

MDC_SM2S - Subset SIMD Mode 2 Message Descriptor Control Field

Source: BSpec
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
00h	SIMD8	SIMD8
01h	Reserved	Ignored

Subset SIMD Mode 3 Message Descriptor Control Field

MDC_SM3S - Subset SIMD Mode 3 Message Descriptor Control Field

Source: BSpec
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description
00h	SIMD4x2	SIMD4x2
01h	Reserved	Ignored
02h	SIMD8	SIMD8
03h	Reserved	Ignored

Subspan Render Target Message Header Control

MHC_RT_SUBSPAN - Subspan Render Target Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Y** Format: U16 Y coordinate for upper-left pixel of this subspan
	15:0	**X** Format: U16 X coordinate for upper-left pixel of this subspan

Surface Binding Table Index Message Descriptor Control Field

MDC_BTS - Surface Binding Table Index Message Descriptor Control Field

Source:	BSpec
Size (in bits):	8
Default Value:	0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Format: Enumeration Specifies the Binding Table index for the message, which must be a Surface State Model.

Value	Name	Description
00h-0EFh	BTS	Index of Binding Table State Surfaces
F0h-0FBh	Reserved	Reserved for future use
0FCh	SSO	Specifies a Surface State Offset supplied by the extended message descriptor
Others	Reserved	Ignored

Surface or Stateless Binding Table Index Message Descriptor Control Field

MDC_BTS_A32 - Surface or Stateless Binding Table Index Message Descriptor Control Field

Source: BSpec
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Format: Enumeration Specifies the surface for the message, either Surface State Model or Stateless.

Value	Name	Description
00h-0EFh	BTS	Index of Binding Table State Surfaces
F0h-0FBh	Reserved	Reserved for future use
0FCh	SSO	Specifies a Surface State Offset supplied by the extended message descriptor
0FFh	A32_A64	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).
Others	Reserved	Ignored

Restriction
Restriction : When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Surface Pitch Message Header Control

MHC_PITCH - Surface Pitch Message Header Control

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Format: Ignore Ignored
	15:0	**Surface Pitch** Format: U16 Specifies the surface width (pitch) to be used for Stateless messages. The Surface Pitch is specified as (#items - 1). The items are specified in the message, and can be bytes, Dwords, or Qwords. The target element address offset is: BufferBaseAddress + U * (SurfacePitch+1) where U is the 32-bit element offset delivered in message address payload.

Surface Pixel Mask Message Header

MH1_BTS_PSM - Surface Pixel Mask Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0..6	223:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

SW Generated BINDING_TABLE_STATE

SW Generated BINDING_TABLE_STATE

Source: BSpec
Size (in bits): 32
Default Value: 0x00000000

Description
The binding table binds surfaces to logical resource indices used by shaders and other compute engine kernels. It is stored as an array of up to 256 elements, each of which contains one dword as defined here. The start of each element is spaced one dword apart.
The first element of the binding table is aligned to a 64-byte boundary.
Binding table indexes beyond 256 will automatically be mapped to entry 0 by the HW, w/ the exception of any messages which support the special indexes 240 through 255, inclusive.

DWord	Bit	Description
0	31:6	**Surface State Pointer** Format: SurfaceStateOffset[31:6] This 64-byte aligned address points to a surface state block. This pointer is relative to the **Surface State Base Address**
	5	**Reserved** Format: MBZ
	4:0	**Reserved** Format: MBZ

SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTW_ZMA8 - SZ OM S0A SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1792

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_ZMA8 - SZ OM S0A SIMD8 Render Target Data Payload

6.0-6.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth

SZ OM S0A SIMD16 Render Target Data Payload

MDP_RTW_ZMA16 - SZ OM S0A SIMD16 Render Target Data Payload

Source:	BSpec
Size (in bits):	3328
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_ZMA16 - SZ OM S0A SIMD16 Render Target Data Payload		
7.0-8.7	511:0	**Blue** Format: **MDP_DW_SIMD16** Slots [15:0] Blue
9.0-10.7	511:0	**Alpha** Format: **MDP_DW_SIMD16** Slots [15:0] Alpha
11.0-12.7	511:0	**Source Depth** Format: **MDP_DW_SIMD16** Slots [15:0] Source Depth

SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_ZM8DS - SZ OM SIMD8 Dual Source Render Target Data Payload

Source: BSpec

Size (in bits): 2560

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_ZM8DS - SZ OM SIMD8 Dual Source Render Target Data Payload		
5.0-5.7	255:0	**Src1 Red** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Red
6.0-6.7	255:0	**Src1 Green** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha
9.0-9.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] or [15:8] of Source Depth

SZ OM SIMD8 Render Target Data Payload

MDP_RTW_ZM8 - SZ OM SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1536

Default Value: 0x00000000, 0x00000000, 0x00000000,

SZ OM SIMD16 Render Target Data Payload

MDP_RTW_ZM16 - SZ OM SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 2816

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_ZM16 - SZ OM SIMD16 Render Target Data Payload		
5.0-5.7	255:0	**Blue[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Blue
6.0-6.7	255:0	**Blue[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
7.0-7.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
8.0-8.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha
9.0-9.7	255:0	**Source Depth[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth
10.0-10.7	255:0	**Source Depth[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Source Depth

SZ S0A SIMD8 Render Target Data Payload

MDP_RTW_ZA8 - SZ S0A SIMD8 Render Target Data Payload

Source: BSpec

Size (in bits): 1536

Default Value: 0x00000000, 0x00000000, 0x00000000,

SZ S0A SIMD16 Render Target Data Payload

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 3072

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

5.0-5.7	255:0	**Green[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Green
6.0-6.7	255:0	**Blue[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Blue
7.0-7.7	255:0	**Blue[15:7]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
8.0-8.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
9.0-9.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha
10.0-10.7	255:0	**Source Depth[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth
11.0-11.7	255:0	**Source Depth[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Source Depth

SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTW_Z8DS - SZ SIMD8 Dual Source Render Target Data Payload

Source:	BSpec
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000, 0x00000000,

MDP_RTW_Z8DS - SZ SIMD8 Dual Source Render Target Data Payload

5.0-5.7	255:0	**Src1 Green** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Green
6.0-6.7	255:0	**Src1 Blue** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Format: MDP_DW_SIMD8 Slots[7:0] or [15:8] of Src1 Alpha
8.0-8.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] or [15:8] of Source Depth

SZ SIMD8 Render Target Data Payload

MDP_RTW_Z8 - SZ SIMD8 Render Target Data Payload

Source:	BSpec
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Format: MDP_DW_SIMD8 Slots [7:0] Red
1.0-1.7	255:0	**Green** Format: MDP_DW_SIMD8 Slots [7:0] Green
2.0-2.7	255:0	**Blue** Format: MDP_DW_SIMD8 Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
4.0-4.7	255:0	**Source Depth** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth

SZ SIMD16 Render Target Data Payload

MDP_RTW_Z16 - SZ SIMD16 Render Target Data Payload

Source: BSpec

Size (in bits): 2560

Default Value: 0x00000000, 0x00000000, 0x00000000,

MDP_RTW_Z16 - SZ SIMD16 Render Target Data Payload		
5.0-5.7	255:0	**Blue[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Blue
6.0-6.7	255:0	**Alpha[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Alpha
7.0-7.7	255:0	**Alpha[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Alpha
8.0-8.7	255:0	**Source Depth[7:0]** Format: MDP_DW_SIMD8 Slots [7:0] Source Depth
9.0-9.7	255:0	**Source Depth[15:8]** Format: MDP_DW_SIMD8 Slots [15:8] Source Depth

Thread EOT Message Descriptor

TS_EOT - Thread EOT Message Descriptor

Source:	RenderCS
Size (in bits):	32
Default Value:	0x02000000

End of Thread message is sent to SFID_TS (07h) to end GPGPU and Media threads. The EU send instruction must also set the EOT control (bit 5) of the extended message descriptor.
This message is sent with single register message payload, which is a copy of the R0 thread payload sent with the thread dispatch.

DWord	Bit	Description
0	31:29	**Reserved** Format: MBZ
	28:25	**Message Length** Default Value: 1h One GRF Format: U4
	24:20	**Response Length** Default Value: 0h Zero GRF Format: U5
	19	**Header Present** Format: MBZ
	18:1	**Reserved** Format: MBZ
	0	**Message Type** Default Value: 0h End Thread Format: Opcode End of Thread message opcode

Thread Spawn Message Descriptor

Thread Spawn Message Descriptor

Source: RenderCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ
	19	**Header Present** Format: MBZ **Programming Notes** This bit MBZ for all Thread Spawner messages.
	18:5	**Reserved** Format: MBZ
	4	**Resource Select** This field specifies the resource associated with the action taken by the Opcode. Value / Name / Description / Exists If: 0 / Spawn Child / Spawn a Child Thread / [Opcode] == 'Spawn Thread' 1 / Spawn Root / Spawn a Root Thread / [Opcode] == 'Spawn Thread' 0 / Dereference Resource / The URB Handle is Dereferenced / [Opcode] == 'Dereference Resource' 1 / Keep Resource / The URBHhandle is NOT Dereferenced / [Opcode] == 'Dereference Resource'
	3:2	**Reserved** Format: MBZ
	1	**Requester Type** This field indicates whether the requesting thread is a root thread or a child thread. If it is a root thread, when Opcode is 0, FF managed resources are dereferenced. If it is a child thread and Opcode is 0, no resource is dereferenced; no action is required by the TS. Value / Name: 0 / Root Thread 1 / Child Thread
	0	**Opcode** Indicates the operation performed by the message. A root thread must terminate with a message to TS (Opcode == 0 and EOT == 1).

Thread Spawn Message Descriptor

A child thread should also terminate with such a message. A thread cannot terminate with an Opcode of "spawn thread".

Value	Name	Description
0	Dereference Resource	also used for end of thread
1	Spawn Thread	

TileW SIMD8 Data Control Dword

MDCD_TILEW - TileW SIMD8 Data Control Dword		
Source:	BSpec	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:8	**Reserved** Format: Ignore Ignored
	7:0	**Red** Format: U8 Specifies the value of the red channel to be read or written.

TileW SIMD8 Data Payload

MDP_TILEW_SIMD8 - TileW SIMD8 Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0	31:0	**Red Slot0** Format: MDCD_TileW Specifies the Slot 0 red channel data
0.1	31:0	**Red Slot1** Format: MDCD_TileW Specifies the Slot 1 red channel data
0.2	31:0	**Red Slot2** Format: MDCD_TileW Specifies the Slot 2 red channel data
0.3	31:0	**Red Slot3** Format: MDCD_TileW Specifies the Slot 3 red channel data
0.4	31:0	**Red Slot4** Format: MDCD_TileW Specifies the Slot 4 red channel data
0.5	31:0	**Red Slot5** Format: MDCD_TileW Specifies the Slot 5 red channel data
0.6	31:0	**Red Slot6** Format: MDCD_TileW Specifies the Slot 6 red channel data
0.7	31:0	**Red Slot7** Format: MDCD_TileW Specifies the Slot 7 red channel data

Transpose Message Header

MH_T - Transpose Message Header

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** This field must be a multiple of the Block Width in bytes. Must be DWORD aligned.
1	31:0	**Y Offset** Format: S31 Y offset (in rows) of the upper left corner of the block into the surface. **Programming Notes** This field must be a multiple of the Block Height.
2	31:0	**Block Dimensions** Format: MHC_BDIM The height and width of the block to transpose.
3..7	159:0	**Reserved** Format: Ignore Ignored

Untyped Write Channel Mask Message Descriptor Control Field

MDC_UW_CMASK - Untyped Write Channel Mask Message Descriptor Control Field

Source: BSpec
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3:0	**Mask** Format: Enumeration For untyped surface write messages, indicates which channels are included in the message payload and written to the surface.

Value	Name	Description
00h	RGBA **[Default]**	Red, Green, Blue, and Alpha are included
08h	RGB	Red, Green, and Blue are included
0Ch	RG	Red and Green are included
0Eh	R	Red is included
Others	Reserved	Ignored

Upper Oword Block Data Payload

MDP_OW1U - Upper Oword Block Data Payload		
Source:	BSpec	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Reserved** Format: Ignore Ignored
0.4-0.7	127:0	**Oword** Format: U128 Specifies the upper Oword data element

VC1

VC1		
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:8	**Reserved** Format: MBZ
	7	**Syncmarker Error** This flag indicates missing sync marker SEs coded in the bit-stream.
	6	**Mbmode SE Error** This flag indicates inconsistent Macroblock SEs coded in the bit-stream.
	5	**Transformtype SE Error** This flag indicates inconsistent transform type SEs coded in the bit-stream.
	4	**Coefficient Error** This flag indicates inconsistent Coefficient SEs coded in the bit-stream.
	3	**Motion Vector SE Error** This flag indicates inconsistent Motion Vector SEs coded in the bit-stream.
	2	**Coded Block Pattern CY SE Error** This flag indicates inconsistent CBPCY SEs coded in the bit-stream.
	1	**Mquant Error** This flag indicates inconsistent MQUANT SEs coded in the bit-stream.
	0	**MB Concealment Flag** . Each pulse from this flag indicates one MB is concealed by hardware.

VCS Hardware-Detected Error Bit Definitions

VCS Hardware-Detected Error Bit Definitions

Source: VideoCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: **Value** / **Name** / **Description** 1 / / Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS

VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS

Source: VideoEnhancementCS
Size (in bits): 11
Default Value: 0x00000000

DWord	Bit	Description
0	10:9	**Tiled Resource Mode for Output Frame Surface Base Address** **For Media Surfaces:** This field specifies the tiled resource mode. Value / Name / Description: 0h / TRMODE_NONE / No tiled resource 1h / TRMODE_TILEYF / 4KB tiled resources 2h / TRMODE_TILEYS / 64KB tiled resources 3h / Reserved /
	8	**Memory Compression Mode** Description: Distinguishes Vertical from Horizontal compression. Value / Name: 0 / Horizontal Compression Mode 1 / Vertical Compression Mode
	7	**Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface.
	6:1	**Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**

VEBOX_ACE_LACE_STATE

VEBOX_ACE_LACE_STATE

Source: VideoEnhancementCS

Size (in bits): 416

Default Value: 0x00000068, 0x4C382410, 0x9C887460, 0xEBD8C4B0, 0x604C3824, 0xB09C8874, 0x0000D8C4, 0x04000400, 0x04000400, 0x04000400, 0x04000400, 0x04000400, 0x00000400

This state structure contains the IECP State Table Contents for ACE state.

DWord	Bit	Description
0	31:16	**Min_ACE_luma** Format: U16
	15:14	**LACE Single Histogram Set** This bit tells LACE which frames will be included in the histogram when the Deinterlacer is enabled.

Value	Name	Description
00b	Current	The histogram includes only the current frame.
01b	Previous	The histogram includes only the previous frame.
10b	Current + Previous	The histogram includes pixels from both the current and previous frame.
11b	Previous + Current	The histogram includes the previous frame followed by the current frame.

Programming Notes
When the Deinterlacer is disabled, this field must be 00b. If **DI Output Frames** is set to only output a single field then the histogram can not be collected on the disabled field. This Field must be set to 00b when DN/DI First Frame is set to 1

	13	**LACE Histogram Size**

Value	Name
0	128-bin histogram
1	256-bin histogram

	12	**LACE Histogram Enable** Default Value: 0 This bit enables the collection of LACE histogram data. If this bit is 0 then only the ACE histogram will be collected.
	11:7	**Reserved** Format: MBZ

VEBOX_ACE_LACE_STATE

	6:2	**Skin Threshold** Format: U5 Used for Y analysis (min/max) for pixels which are higher than skin threshold. Value / Name [1,31] 26 **[Default]**
	1	**Reserved** Format: MBZ
	0	**ACE Enable** Format: Enable
1	31:24	**Y3** Default Value: 76 Format: U8 The value of the y_pixel for point 3 in PWL.
	23:16	**Y2** Default Value: 56 Format: U8 The value of the y_pixel for point 2 in PWL.
	15:8	**Y1** Default Value: 36 Format: U8 The value of the y_pixel for point 1 in PWL.
	7:0	**Ymin** Default Value: 16 Format: U8 The value of the y_pixel for point 0 in PWL.
2	31:24	**Y7** Default Value: 156 Format: U8 The value of the y_pixel for point 7 in PWL.

VEBOX_ACE_LACE_STATE

	23:16	**Y6** Default Value: 136 Format: U8 The value of the y_pixel for point 6 in PWL.
	15:8	**Y5** Default Value: 116 Format: U8 The value of the y_pixel for point 5 in PWL.
	7:0	**Y4** Default Value: 96 Format: U8 The value of the y_pixel for point 4 in PWL.
3	31:24	**Ymax** Default Value: 235 Format: U8 The value of the y_pixel for point 11 in PWL.
	23:16	**Y10** Default Value: 216 Format: U8 The value of the y_pixel for point 10 in PWL.
	15:8	**Y9** Default Value: 196 Format: U8 The value of the y_pixel for point 9 in PWL.
	7:0	**Y8** Default Value: 176 Format: U8 The value of the y_pixel for point 8 in PWL.
4	31:24	**B4** Default Value: 96 Format: U8 The value of the bias for point 4 in PWL.

VEBOX_ACE_LACE_STATE

	23:16	**B3** Default Value: 76 Format: U8 The value of the bias for point 3 in PWL.
	15:8	**B2** Default Value: 56 Format: U8 The value of the bias for point 2 in PWL.
	7:0	**B1** Default Value: 36 Format: U8 The value of the bias for point 1 in PWL.
5	31:24	**B8** Default Value: 176 Format: U8 The value of the bias for point 8 in PWL.
	23:16	**B7** Default Value: 156 Format: U8 The value of the bias for point 7 in PWL.
	15:8	**B6** Default Value: 136 Format: U8 The value of the bias for point 6 in PWL.
	7:0	**B5** Default Value: 116 Format: U8 The value of the bias for point 5 in PWL.
6	31:16	**Reserved** Format: MBZ

VEBOX_ACE_LACE_STATE

	15:8	**B10** Default Value: 216 Format: U8 The value of the bias for point 10 in PWL.
	7:0	**B9** Default Value: 196 Format: U8 The value of the bias for point 9 in PWL.
7	31:27	**Reserved** Format: MBZ
	26:16	**S1** Default Value: 1024 Format: U1.10 The value of the slope for point 1 in PWL The default is 1024/1024
	15:11	**Reserved** Format: MBZ
	10:0	**S0** Default Value: 1024 Format: U1.10 The value of the slope for point 0 in PWL The default is 1024/1024
8	31:27	**Reserved** Format: MBZ
	26:16	**S3** Default Value: 1024 Format: U1.10 The value of the slope for point 3 in PWL The default is 1024/1024
	15:11	**Reserved** Format: MBZ

VEBOX_ACE_LACE_STATE

	10:0	**S2** Default Value: 1024 Format: U1.10 The value of the slope for point 2 in PWL The default is 1024/1024
9	31:27	**Reserved** Format: MBZ
	26:16	**S5** Default Value: 1024 Format: U1.10 The value of the slope for point 5 in PWL The default is 1024/1024
	15:11	**Reserved** Format: MBZ
	10:0	**S4** Default Value: 1024 Format: U1.10 The value of the slope for point 4 in PWL The default is 1024/1024
10	31:27	**Reserved** Format: MBZ
	26:16	**S7** Default Value: 1024 Format: U1.10 The value of the slope for point 7 in PWL The default is 1024/1024
	15:11	**Reserved** Format: MBZ

VEBOX_ACE_LACE_STATE

	10:0	**S6** Default Value: 1024 Format: U1.10 The default is 1024/1024
11	31:27	**Reserved** Format: MBZ
	26:16	**S9** Default Value: 1024 Format: U1.10 The value of the slope for point 9 in PWL The default is 1024/1024
	15:11	**Reserved** Format: MBZ
	10:0	**S8** Default Value: 1024 Format: U1.10 The value of the slope for point 8 in PWL The default is 1024/1024
12	31:16	**Max_ACE_luma** Format: U16 The maximum luma for which ACE correction will be used.
	15:11	**Reserved** Format: MBZ
	10:0	**S10** Default Value: 1024 Format: U1.10 The value of the slope for point 10 in PWL.

VEBOX_ALPHA_AOI_STATE

VEBOX_ALPHA_AOI_STATE

Source: VideoEnhancementCS
Size (in bits): 96
Default Value: 0x00000000, 0x00030000, 0x00030000

This state structure contains the IECP State Table Contents for Fixed Alpha State and Area of Interest State.

DWord	Bit	Description
0	31:18	**Reserved** Format: MBZ
	17	**Full Image Histogram** Default Value: 0 Format: Enable Used to ignore the area of interest for a histogram across the full image. This applies to all statistics that are affected by AOI (Area of Interest).
	16	**Alpha from State Select** Format: U1 Enumerated type Value / Name: 0 – alpha is taken from message 1 – alpha is taken from state **Programming Notes** If the input format does not have alpha available and the output format provides alpha, this bit should be set to 1. This should be 0 when Alpha Plane Enable is 1.
	15:0	**Color Pipe Alpha** Format: U16 **Programming Notes** The 8 MSB of this field will be used for output formats that have 8-bits of alpha.
1	31:30	**Reserved** Format: MBZ
	29:16	**AOI Max X** Default Value: 3 Format: U14

VEBOX_ALPHA_AOI_STATE

		Area of Interest Minimum X - The ACE histogram and Skin Tone Detection statistic gathering will occur within the MinX/MinY to MaxX/MaxY area (inclusive). AOI must intersect the frame such that at least 1 pixel is in the AOI. The Area of Interest applies to the RGB Histogram and the White/Gray point sums as well. **Programming Notes** **This value must be a multiple of 4 minus 1.**
	15:14	**Reserved** Format: MBZ
	13:0	**AOI Min X** Default Value: 0 Format: U14 **Programming Notes** **This value must be a multiple of 4.**
2	31:30	**Reserved** Format: MBZ
	29:16	**AOI Max Y** Default Value: 3 Format: U14 **Programming Notes** **This value must be a multiple of 4 minus 1.**
	15:14	**Reserved** Format: MBZ
	13:0	**AOI Min Y** Default Value: 0 Format: U14 **Programming Notes** **This value must be a multiple of 4.**

VEBOX_CAPTURE_PIPE_STATE

VEBOX_CAPTURE_PIPE_STATE

Source: VideoEnhancementCS

Size (in bits): 224

Default Value: 0x0511FF23, 0xAA64AFAA, 0xE6FD4000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This command contains variables for controlling Demosaic and the White Balance Statistics.

DWord	Bit	Description
0	31:30	**Reserved** Format: MBZ
	29:24	**Good Pixel Threshold** Format: U6 The difference threshold between adjacent pixels for a pixel to be considered "good". Value: 5h — Name: **[Default]**
	23	**Reserved** Format: MBZ
	22:20	**Shift Min Cost** Default Value: 1h Format: U3 The amount to shift the H2/V2 versions of min_cost.
	19:16	**Green Imbalance Threshold** Default Value: 1h Format: U4
	15:8	**Average Color Threshold** Format: U8 The threshold between two colors in a pixel for the Avg interpolation to be considered. Value: FFh — Name: **[Default]** **Programming Notes** Must be set to 255.
	7:6	**Reserved** Format: U2

VEBOX_CAPTURE_PIPE_STATE

	5:0	**Good Pixel Neighbor Threshold** Default Value: 23h Format: U6 Number of comparisons with neighbor pixels which pass before a pixel is considered good.
1	31:28	**Scale For Min Cost** Default Value: Ah The amount to scale the min_cost difference during the confidence check.
	27:24	**Good Intesity Threshold** Default Value: Ah Format: U4
	23:16	**Bad Color Threshold 1** Default Value: 64h Format: U8 Color value threshold used during the bad pixel check.
	15:8	**Bad Color Threshold 2** Default Value: AFh Format: U8 Color value threshold used during the bad pixel check.
	7:4	**Number Big Pixel Threshold** Default Value: Ah Format: U4 Number of comparisons with neighbor pixels which pass before a pixel is considered good.
	3:0	**Bad Color Threshold 3** Default Value: Ah Format: U4 Color value threshold used during the bad pixel check.
2	31:24	**Y Bright Value** Default Value: E6h The whitepoint threshold percentile in the Y histogram. Any pixel with Y value above this could be a whitepoint. This is the larger of the calculated Ybright value and the Ythreshold value, which is the minimum Y required to be considered a white point. **Programming Notes**

VEBOX_CAPTURE_PIPE_STATE

		"00000000" is appended to the LSBs before comparing with Y.
	23:16	**Y Outlier Value** Default Value: FDh The outlier threshold percentile in the Y histogram. Any pixel with Y value above this either clipped or an outlier in the image. These points will not be included in the white patch calculation. **Programming Notes** "00000000" is appended to the LSBs before comparing with Y.
	15:8	**UV Threshold Value** The value denotes the maximum threshold of the ratio between U+V to Y can have to be considered a gray point. Value: [255,0] — Name: — Description: Encode a value from 255/256 to 0/256 Value: 64 — Name: **[Default]** — Description: 0.25 * 255 = 64
	7	**Black Point Offset Red MSB**
	6	**Black Point Offset Green Top MSB**
	5	**Black Point Offset Blue MSB**
	4	**Black Point Offset Green Bottom MSB**
	3	**RGB Histogram Enable** Enables the collection of RGB Histograms for Auto-white balance correction and other uses. **Programming Notes** This bit can be set without White Balance enable being set.
	2	**Vignette Correction Format** Defines what shift should be assumed for the **Vignette** Correction input values: Value: 0 — Name: U8.8 Value: 1 — Name: U4.12
	1	**Black Point Correction Enable** Format: Enable
	0	**White Balance Correction Enable** Format: Enable **Programming Notes** RGB Histogram enable must be set if this bit is set.
3	31:16	**Black Point Offset Red** Default Value: 0 Format: U16

VEBOX_CAPTURE_PIPE_STATE

		Value subtracted from Red pixels of Bayer pattern - combined with MSB to form a 2's complement signed number.
	15:0	**Black Point Offset Green Top** Default Value: 0 Format: U16 Value subtracted from the top Green pixels of Bayer pattern (X=1, Y=0 for Bayer Pattern #1) - combined with MSB to form a 2's complement signed number.
4	31:16	**Black Point Offset Blue** Default Value: 0 Format: U16 Value subtracted from Blue pixels of Bayer pattern - Combine with MSB to form a 2's complement signed number.
	15:0	**Black Point Offset Green Bottom** Default Value: 0 Format: U16 Value subtracted from the bottom Green pixels of Bayer pattern (X=0, Y=1 for Bayer Pattern #1) - combined with MSB to form a 2's complement signed number.
5	31:16	**White Balance Red Correction** Format: U4.12 The correction factor multiplied by the Red pixels of the Bayer pattern.
	15:0	**White Balance Green Top Correction** Format: U4.12 The correction factor multiplied by the top Green pixels of the Bayer pattern(X=1, Y=0 for Bayer Pattern #1).
6	31:16	**White Balance Blue Correction** Format: U4.12 The correction factor multiplied by the Blue pixels of the Bayer pattern.
	15:0	**White Balance Green Bottom Correction** Format: U4.12 The correction factor multiplied by the bottom Green pixels of the Bayer pattern (X=0, Y=1 for Bayer Pattern #1)

VEBOX_CCM_STATE

VEBOX_CCM_STATE		
Source:	VideoEnhancementCS	
Size (in bits):	288	
Default Value:	0x00000475, 0x00000AE8, 0x00000047, 0x00000022, 0x0001FFCC, 0x00000D23, 0x000000A8, 0x0001FFF4, 0x00000D6A	
This state structure contains the IECP State Table Contents for the Color Correction Matrix State.		
DWord	**Bit**	**Description**
0	31	**Color Correction Matrix Enable** Format: Enable This bit enables the Color Correction Matrix. **Programming Notes** Single Pipe IECP Enable must also be set if this bit is enabled.
	30:17	**Reserved** Format: MBZ
	16:0	**C1** Default Value: 000475h = 1141/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
1	31:17	**Reserved** Format: MBZ
	16:0	**C0** Default Value: 000AE8h = 2792/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
2	31:17	**Reserved** Format: MBZ
	16:0	**C3** Default Value: 000047h = 71/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
3	31:17	**Reserved** Format: MBZ

VEBOX_CCM_STATE

	16:0	**C2** Default Value: 000022h = 34/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
4	31:17	**Reserved** Format: MBZ
	16:0	**C5** Default Value: 1FFFCCh = -52/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
5	31:17	**Reserved** Format: MBZ
	16:0	**C4** Default Value: 000D23h = 3363/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
6	31:17	**Reserved** Format: MBZ
	16:0	**C7** Default Value: 0000A8h = 168/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
7	31:17	**Reserved** Format: MBZ
	16:0	**C6** Default Value: 1FFFF4h = -12/4096 Format: S4.12 Coefficient of 3x3 Transform matrix
8	31:17	**Reserved** Format: MBZ

VEBOX_CCM_STATE		
	16:0	**C8** Default Value: 000D6Ah = 3434/4096 Format: S4.12 Coefficient of 3x3 Transform matrix

VEBOX_Ch_Dir_Filter_Coefficient

VEBOX_Ch_Dir_Filter_Coefficient		
Source:	BSpec	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:56	**Filter Coefficient[7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	55:48	**Filter Coefficient[6]** Format: S1.6 2's Complement **Range:** [-2, +2)
	47:40	**Filter Coefficient[5]** Format: S1.6 2's Complement **Range:** [-2, +2)
	39:32	**Filter Coefficient[4]** Format: S1.6 2's Complement **Range:** [-2, +2)
	31:24	**Filter Coefficient[3]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Filter Coefficient[2]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Filter Coefficient[1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**Filter Coefficient[0]** Format: S1.6 2's Complement **Range:** [-2, +2)

VEBOX_CSC_STATE

VEBOX_CSC_STATE

Source: VideoEnhancementCS

Size (in bits): 384

Default Value: 0x00010000, 0x00000000, 0x00000000, 0x00000000, 0x00010000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00010000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the IECP State Table Contents for CSC state.

DWord	Bit	Description
0	31	**Transform Enable** Format: Enable
	30	**YUV_Channel_Swap** Default Value: 0 Format: Enable This bit should only be used with R8G8B8A8 output formats. When this bit is set, the YUV channels are swapped into the output RGB channels as shown in the following table to support B8G8R8A8 output on surface format R8G8B8A8: (, YUV_Channel_Swap) (, 0 --> 1) (Y, R --> B) (U, G --> G) (V, B --> R) **Programming Notes** This bit is to provide support for B8G8R8A8 output even though it is not supported in the surface format.
	29:19	**Reserved** Format: MBZ
	18:0	**C0** Default Value: 10000h or 1.0 Format: S2.16 2's complement Transform coefficient.
1	31:19	**Reserved** Format: MBZ

VEBOX_CSC_STATE

	18:0	**C1** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
2	31:19	**Reserved** Format: MBZ
	18:0	**C2** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
3	31:19	**Reserved** Format: MBZ
	18:0	**C3** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
4	31:19	**Reserved** Format: MBZ
	18:0	**C4** Default Value: 10000h or 1.0 Format: S2.16 2's complement Transform coefficient.
5	31:19	**Reserved** Format: MBZ
	18:0	**C5** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
6	31:19	**Reserved** Format: MBZ

VEBOX_CSC_STATE

	18:0	**C6** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
7	31:19	**Reserved** Format: MBZ
	18:0	**C7** Default Value: 0 Format: S2.16 2's complement Transform coefficient.
8	31:19	**Reserved** Format: MBZ
	18:0	**C8** Default Value: 10000h or 1.0 Format: S2.16 2's complement Transform coefficient. The offset value is multiplied by 2 before being added to the output.
9	31:16	**Offset Out 1** Default Value: 0 Format: S15 2's Complement Offset in for Y/R. The offset value is multiplied by 2 before being added to the output.
	15:0	**Offset in 1** Default Value: 0 Format: S15 2's Complement Offset in for Y/R. The offset value is multiplied by 2 before being added to the output.
10	31:16	**Offset Out 2** Default Value: 0 Format: S15 2's Complement Offset out for U/G. The offset value is multiplied by 2 before being added to the output.
	15:0	**Offset in 2** Default Value: 0 Format: S15 2's Complement Offset out for U/G. The offset value is multiplied by 2 before being added to the output.

VEBOX_CSC_STATE		
11	31:16	**Offset Out 3** Default Value: 0 Format: S15 2's Complement Offset out for V/B. The offset value is multiplied by 2 before being added to the output.
	15:0	**Offset in 3** Default Value: 0 Format: S15 2's Complement Offset out for V/B. The offset value is multiplied by 2 before being added to the output.

VEBOX_DNDI_STATE

VEBOX_DNDI_STATE

Source: VideoEnhancementCS

Size (in bits): 576

Default Value: 0x00000800, 0x00000000, 0x00008000, 0x00010000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000254, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x005064A5

This state table is used by the *Denoise and Deinterlacer functions*.

- DW0 to 2 are for Temporal Denoise
- DW3 is for global noise estimate and hot pixel detection
- DW4 is for Chroma Denoise
- DW5 to 11 are for 5x5 spatial denoise
- DW12 to 17 are for Deinterlacer
- DW18 to 24 Reserved

DWord	Bit	Description
0	31:20	**Denoise STAD Threshold** Format: U12 Threshold for denoise sum of temporal absolute differences.
	19:12	**Denoise Maximum History** Format: U8 Maximum allowed value for denoise history. Value: [128,240] — Name:
	11:8	**Denoise History increase** Format: U4 Amount that denoise_history is increased by. MAX:15 Value: 8h — Name: **[Default]** Value: 15 — Description: Maximum Allowed
	7:5	**Reserved** Format: MBZ
	4:0	**Denoise Moving Pixel Threshold** Format: U5 Value — Name

VEBOX_DNDI_STATE

		[0,16]
1	31:20	**Denoise ASD Threshold** Format: U12 Threshold for denoise absolute sum of differences. Value: [0,1023]; Name:
	19:10	**Temporal Difference Threshold** Format: U10 **Programming Notes** 0 <(**Temporal Difference Threshold** - **Low Temporal Difference Threshold**) < = 256 except when both thresholds are set to 0.
	9:0	**Low Temporal Difference Threshold** Format: U10 **Programming Notes** 0 <(**Temporal Difference Threshold** - **Low Temporal Difference Threshold**) < = 256 except when both thresholds are set to 0.
2	31:29	**Reserved** Format: MBZ
	28	**Progressive DN** Format: Enable Indicates that the denoise algorithm should assume progressive input when filtering neighboring pixels. **This bit must be set if the input to Denoise is RGB.** Value 0: DN assumes interlaced video and filters alternate lines together Value 1: DN assumes progressive video and filters neighboring lines together **Programming Notes** **DI Enable** must be disabled when this field is enabled.
	27:16	**Denoise Threshold for Sum of Complexity Measure** Format: U12
	15:10	**Initial Denoise History** Default Value: 32 Format: U6 Initial value for Denoise history for both Luma and Chroma

VEBOX_DNDI_STATE

9:0 Reserved

Format:	MBZ

3 — 31:28 Hot Pixel Count

Format:	U4

Number of neighboring pixels different more than **Hot Pixel Threshold** before a pixel is considered hot.

Value	Name
[0,8]	

Programming Notes
0 will cause all pixels to be considered hot and will perform a median filter on the entire image.

27:20 Hot Pixel Threshold

Format:	U8

Threshold for a difference from the value of a neighboring pixel. Is shifted up to 16-bits before compare.

19:12 Block Noise Estimate Edge Threshold

Format:	U8

Threshold for detecting an edge in block noise estimate.

Value	Name	Description
16	**[Default]**	
255		Maximium Value

11:0 Block Noise Estimate Noise Threshold

Format:	U12

Threshold for noise maximum/minimum.

Value	Name
[0, 511]	

4 — 31:24 Reserved

Format:	MBZ

23:16 Chroma Denoise STAD Threshold

Format:	U8

Threshold for denoise sum of temporal absolute differences.

15:13 Reserved

Format:	MBZ

12 Chroma Denoise Enable

Value	Name	Description

VEBOX_DNDI_STATE

		1 | The U and V chroma channels will be denoise filtered. 0 | The U and V channels will be passed to the next stage after DN unchanged.
	11:6	**Chroma Temporal Difference Threshold** Format: U6 **Programming Notes** 0 < (**Chroma Temporal Difference Threshold** - **Chroma Low Temporal Difference Threshold**)<=16 (Larger than 0 and less than or equal to 16)
	5:0	**Chroma Low Temporal Difference Threshold** Format: U6 **Programming Notes** 0 < (**Chroma Temporal Difference Threshold** - **Chroma Low Temporal Difference Threshold**) <= 16
5	31:30	**Reserved** Format: MBZ
	29:25	**Dn_Wr5** th4 <= (Weight to be applied to pixels with differences in pixel value)
	24:20	**Dn_Wr4** th3 <= (Weight to be applied to pixels with differences in pixel value) < th4
	19:15	**Dn_Wr3** th2 <= (Weight to be applied to pixels with differences in pixel value) < th3
	14:10	**Dn_Wr2** th1 <= (Weight to be applied to pixels with differences in pixel value) < th2
	9:5	**Dn_Wr1** th0 <= (Weight to be applied to pixels with differences in pixel value) < th1
	4:0	**Dn_Wr0** (Weight to be applied to pixels with differences in pixel value) < th0
6	31:29	**Reserved** Format: MBZ
	28:16	**Dn_thmax** Maximum threshold value
	15:13	**Reserved** Format: MBZ
	12:0	**Dn_thmin** Minimum threshold value

VEBOX_DNDI_STATE		
7	31:29	**Reserved** Format: MBZ
	28:16	**Dn_prt5**
	15:13	**Reserved** Format: MBZ
	12:0	**Dn_dyn_thmin** Minimum Dynamic threshold value.
8	31:29	**Reserved** Format: MBZ
	28:16	**Dn_prt4** Multiplied by thrscale and then used as the threshold for comparing the pixel differences.
	15:13	**Reserved** Format: MBZ
	12:0	**Dn_prt3**
9	31:29	**Reserved** Format: MBZ
	28:16	**Dn_prt2**
	15:13	**Reserved** Format: MBZ
	12:0	**Dn_prt1**
10	31:29	**Reserved** Format: MBZ
	28:16	**Dn_prt0**
	15	**Reserved** Format: MBZ
	14:10	**Dn_wd22** Weight to be applied to the 4 pixels that are at X±2 and Y±2
	9:5	**Dn_wd21** Weight to be applied to the 4 pixels that are at X±1 and Y±2
	4:0	**Dn_wd20** Weight to be applied to the 2 pixels that are at X and Y±2
11	31:30	**Reserved** Format: MBZ
	29:25	**Dn_wd12** Weight to be applied to the 4 pixels that are at X±2 and Y±1
	24:20	**Dn_wd11** Weight to be applied to the 4 pixels that are at X±1 and Y±1

VEBOX_DNDI_STATE

	19:15	**Dn_wd10** Weight to be applied to the 2 pixels that are at X and Y±1
	14:10	**Dn_wd02** Weight to be applied to the 2 pixels that are at X±2 and Y
	9:5	**Dn_wd01** Weight to be applied to the 2 pixels that are at X±1 and Y
	4:0	**Dn_wd00** Weight to be applied to the 1 pixels that are at X and Y
12	31:13	**Reserved** Format: MBZ
	12:10	**STMM C2** Format: U3 Bias for divisor in STMM equation. Value / Name / Description: [0,7] / / Representing values [1,8]
	9:6	**Content Adaptive Threshold Slope** Format: U4 Determines the slope of the Content Adaptive Threshold. Value / Name / Description: 9 / **[Default]** / CAT_slope value = 10 **Programming Notes**: +1 added internally to get CAT_slope.
	5:2	**SAD Tight Threshold** Default Value: 5 Format: U4
	1:0	**Smooth MV Threshold** Format: U2
13	31	**STMM Blending Constant Select** Format: U1 Value / Name / Description: 0 / / Use the blending constant for small values of STMM for stmm_md_th 1 / / Use the blending constant for large values of STMM for stmm_md_th
	30:24	**Blending constant across time for large values of STMM** Format: U7

VEBOX_DNDI_STATE

	23:16	**Blending constant across time for small values of STMM** Format: U8
	15:14	**Reserved** Format: MBZ
	13:8	**Multiplier for VECM** Format: U6 Determines the strength of the vertical edge complexity measure.
	7:0	**Maximum STMM** Format: U8 Largest allowed STMM in blending equations.
14	31:24	**Minimum STMM** Format: U8 Smallest allowed STMM in blending equations
	23:22	**STMM Shift Down** Format: U2 Amount to shift STMM down (quantize to fewer bits) Value / Name: 0 – Shift by 4 1 – Shift by 5 2 – Shift by 6 3 – Reserved
	21:20	**STMM Shift Up** Format: U2 Amount to shift STMM up (set range). Value / Name: 0 – Shift by 6 1 – Shift by 7 2 – Shift by 8 3 – Reserved
	19:16	**STMM Output Shift** Format: U4 Value / Name: [0,16]

VEBOX_DNDI_STATE

		Programming Notes The value of this field must satisfy the following equation: stmm_max - stmm_min = 2 ^ stmm_output_shift
	15:8	**SDI Threshold** Format: U8 Threshold for angle detection in SDI algorithm.
	7:0	**SDI Delta** Format: U8 Delta value for angle detection in SDI algorithm.
15]	31:24	**SDI Fallback Mode 1 T1 Constant** Format: U8
	23:16	**SDI Fallback Mode 1 T2 Constant** Format: U8
	15:8	**SDI Fallback Mode 2 Constant (Angle2x1)** Format: U8
	7:0	**FMD Temporal Difference Threshold** Format: U8
16	31:24	**FMD #1 Vertical Difference Threshold** Format: U8
	23:16	**FMD #2 Vertical Difference Threshold** Format: U8
	15:14	**CAT Threshold** Default Value: 0 Format: U2
	13:8	**FMD Tear Threshold** Format: U6
	7	**MCDI Enable** Use Motion Compensated Deinterlace algorithm. **Programming Notes** This bit is Ignored if DI Enable is off.
	6:4	**Reserved** Format: MBZ

VEBOX_DNDI_STATE

3 — DN/DI Top First

Format:	Enable

Indicates the top field is first in sequence, otherwise bottom is first.

Value	Name	Description
0		Bottom field occurs first in sequence
1		Top field occurs first in sequence

2:0 — Reserved

Format:	MBZ

17 — 31:26 Reserved

Format:	MBZ

25:23 Reserved

Format:	MBZ

22:19 Neighbor Pixel Threshold

Default Value:	10
Format:	U4

18 Reserved

Format:	MBZ

17:16 Progressive Cadence Reconstruction For 2nd Field Of Previous Frame

Format:	U2

Value	Name	Description
0	Deinterlace	
1	Put together with previous field in sequence	1st field of previous frame
2	Put together with next field in sequence	1st field of current frame

15:10 MC Pixel Consistency Threshold

Default Value:	25
Format:	U6

9:8 Progressive Cadence Reconstruction for 1st Field of Current Frame

Format:	U2

Value	Name	Description
0	Deinterlace	
1	Put together with previous field in sequence	2nd field of previous frame
2	Put together with next field in sequence	2nd field of current frame

VEBOX_DNDI_STATE		
	7:4	**SAD THB** Default Value: 10 Format: U4
	3:0	**SAD THA** Default Value: 5 Format: U4

VEBOX_Filter_Coefficient

VEBOX_Filter_Coefficient		
Source:	BSpec	
Size (in bits):	8	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	7:0	**2's Complement Filter Coefficient** Format: S1.6 2's Complement **Range:** [-2, +2)

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

Source: VideoEnhancementCS

Size (in bits): 4096

Default Value: 0x00000000, 0x00000000, 0x00000000,

VEBOX_FRONT_END_CSC_STATE

VEBOX_FRONT_END_CSC_STATE		
Source:	VideoEnhancementCS	
Size (in bits):	384	
Default Value:	0x00010000, 0x00000000, 0x00000000, 0x00000000, 0x00010000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00010000, 0x00000000, 0x00000000, 0x00000000	
This state structure contains the IECP State Table Contents for Front-end CSC state.		
DWord	**Bit**	**Description**
0	31	**Front End CSC Transform Enable** Format: Enable **Programming Notes** **Single Pipe IECP Enable** must also be set if this is enabled.
	30:19	**Reserved** Format: MBZ
	18:0	**FECSC C0: Transform coefficient** Default Value: 10000h or 1.0 Format: S2.16
1	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C1: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
2	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C2: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
3	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C3: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
4	31:19	**Reserved** Format: MBZ

VEBOX_FRONT_END_CSC_STATE

	18:0	**FECSC C4: Transform coefficient** Default Value: 10000h or 1.0 Format: S2.16
5	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C5: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
6	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C6: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
7	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C7: Transform coefficient** Default Value: 0 or 0.0 Format: S2.16
8	31:19	**Reserved** Format: MBZ
	18:0	**FECSC C8: Transform coefficient** Default Value: 10000h or 1.0 Format: S2.16
9	31:16	**FEC SC Offset out 1: Offset out for Y/R** Default Value: 0 Format: S15 The offset value is multiplied by 2 before being added to the output.
	15:0	**FEC SC Offset in 1: Offset in for Y/R** Default Value: 0 Format: S15 The offset value is multiplied by 2 before being added to the output.

<table>
<tr><th colspan="3">VEBOX_FRONT_END_CSC_STATE</th></tr>
<tr><td rowspan="2">10</td><td>31:16</td><td>FEC SC Offset out 2: Offset out for U/G<table><tr><td>Default Value:</td><td>0</td></tr><tr><td>Format:</td><td>S15</td></tr></table>The offset value is multiplied by 2 before being added to the output.</td></tr>
<tr><td>15:0</td><td>FEC SC Offset in 2: Offset out for U/G<table><tr><td>Default Value:</td><td>0</td></tr><tr><td>Format:</td><td>S15</td></tr></table>The offset value is multiplied by 2 before being added to the output.</td></tr>
<tr><td rowspan="2">11</td><td>31:16</td><td>FEC SC Offset out 3: Offset out for V/B<table><tr><td>Default Value:</td><td>0</td></tr><tr><td>Format:</td><td>S15</td></tr></table>The offset value is multiplied by 2 before being added to the output.</td></tr>
<tr><td>15:0</td><td>FEC SC Offset in 3: Offset out for V/B<table><tr><td>Default Value:</td><td>0</td></tr><tr><td>Format:</td><td>S15</td></tr></table>The offset value is multiplied by 2 before being added to the output.</td></tr>
</table>

VEBOX_GAMUT_STATE

VEBOX_GAMUT_STATE		
Source:	VideoEnhancementCS	
Size (in bits):	1216	
Default Value:	0x01B40000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x09050201, 0x412A1A10, 0x00BB8860, 0x3526170D, 0x8B725B47, 0x00DFC1A5, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x654F371E, 0x00000000, 0x00EDDBC8, 0x21140A03, 0x755C4331, 0x00D7B493, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0CD2911F, 0xB0000334, 0x00000000	
DWord	**Bit**	**Description**
0	31:25	**Reserved** Format: MBZ
	24:16	**A(r)** Default Value: 436 Format: U9 Gain_factor_R (default: 436, preferred range: 256-511)
	15	**Global Mode Enable** The gain factor derived from state CM(w) Value: 0 — Advance Mode Value: 1 — Basic Mode
	14:10	**Reserved** Format: MBZ
	9:0	**CM(w)** Format: U10 WeightingFactorForGain_factor (only enabled when the GlobalModeEnable is on)
1	31:26	**Reserved** Format: MBZ
	25:16	**CM(s)** Format: U2.8 AccurateColorComponentScaling (default: 640/256, preferred range: [512-1023]/256) The default is 640/256

VEBOX_GAMUT_STATE

	15	**Reserved** Format: MBZ
	14:8	**A(g)** Format: U7 Gain_factor_G (default: 26/256, preferred range: [26-127]/256) The default is 26/256
	7	**Reserved** Format: MBZ
	6:0	**A(b)** Format: U7 Gain_factor_B (default: 26/256, preferred range: [26-127]/256) The default is 26/256
2	31:26	**Reserved** Format: MBZ
	25:16	**R(s)** Format: U2.8 RedScaling (default: 768/256, preferred range: [512-1023]/256) The default is 768/256
	15:8	**CM(i)** Format: U0.8 AccurateColorComponentOffset (default: 192/256, preferred range: [0-192]/256) The default is 192/256
	7:0	**R(i)** Format: U0.8 RedOffset (default: 128/256, preferred range: [0-128]/256) The default is 128/256
3	31	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE

	30:16	**C1** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 1141/4096
	15	**Reserved** Format: MBZ
	14:0	**C0** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 2792/4096
4	31	**Reserved** Format: MBZ
	30:16	**C3** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 71/4096
	15	**Reserved** Format: MBZ
	14:0	**C2** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 34/4096
5	31	**Reserved** Format: MBZ
	30:16	**C5** Format: S2.12 Coefficient of 3x3 Transform matrix The default is -52/4096
	15	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE

	14:0	**C4** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 3663/4096
6	31	**Reserved** Format: MBZ
	30:16	**C7** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 168/4096
	15	**Reserved** Format: MBZ
	14:0	**C6** Format: S2.12 Coefficient of 3x3 Transform matrix The default is -12/4096
7	31:15	**Reserved** Format: MBZ
	14:0	**C8** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 3434/4096
8	31:24	**PWL_Gamma_ Point 4** Default Value: 9 Format: U8 Point 4 for PWL for gamma correction
	23:16	**PWL_Gamma_ Point 3** Default Value: 5 Format: U8 Point 3 for PWL for gamma correction

VEBOX_GAMUT_STATE

	15:8	**PWL_Gamma_ Point 2** Default Value: 2 Format: U8 Point 2 for PWL for gamma correction
	7:0	**PWL_Gamma_ Point 1** Default Value: 1 Format: U8 Point 1 for PWL for gamma correction
9	31:24	**PWL_Gamma_ Point 8** Default Value: 65 Point 8 for PWL for gamma correction
	23:16	**PWL_Gamma_ Point 7** Default Value: 42 Point 7 for PWL for gamma correction
	15:8	**PWL_Gamma_ Point 6** Default Value: 26 Point 6 for PWL for gamma correction
	7:0	**PWL_Gamma_ Point 5** Default Value: 16 Point 5 for PWL for gamma correction
10	31:24	**Reserved** Format: MBZ
	23:16	**PWL_Gamma_ Point 11** Default Value: 187 Format: U8 Point 11 for PWL for gamma correction
	15:8	**PWL_Gamma_ Point 10** Default Value: 136 Format: U8 Point 10 for PWL for gamma correction

VEBOX_GAMUT_STATE

	7:0	**PWL_Gamma_ Point 9** Default Value: 96 Format: U8 Point 9 for PWL for gamma correction
11	31:24	**PWL_Gamma_ Bias_4** Default Value: 53 Format: U8 Bias 4 for PWL for gamma correction
	23:16	**PWL_Gamma_ Bias_3** Default Value: 38 Format: U8 Bias 3 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_2** Default Value: 23 Format: U8 Bias 2 for PWL for gamma correction
	7:0	**PWL_Gamma_ Bias_1** Default Value: 13 Format: U8 Bias 1 for PWL for gamma correction
12	31:24	**PWL_Gamma_ Bias_8** Default Value: 139 Format: U8 Bias 8 for PWL for gamma correction
	23:16	**PWL_Gamma_ Bias_7** Default Value: 114 Format: U8 Bias 7 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_6** Default Value: 91 Format: U8 Bias 6 for PWL for gamma correction

VEBOX_GAMUT_STATE

	7:0	**PWL_Gamma_ Bias_5** Default Value: 71 Format: U8 Bias 5 for PWL for gamma correction
13	31:24	**Reserved** Format: MBZ
	23:16	**PWL_Gamma_ Bias_11** Default Value: 223 Format: U8 Bias 11 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_10** Default Value: 193 Format: U8 Bias 10 for PWL for gamma correction
	7:0	**PWL_Gamma_ Bias_9** Default Value: 165 Format: U8 Bias 9 for PWL for gamma correction
14	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_1** Format: U4.8 Slope 1 for PWL for gamma correction The default is 2560/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_0** Format: U4.8 Slope 0 for PWL for gamma correction The default is 3328/256
15	31:28	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE

	27:16	**PWL_Gamma_ Slope_3** Format: U4.8 Slope 3 for PWL for gamma correction The default is 960/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_2** Format: U4.8 Slope 2 for PWL for gamma correction The default is 1280/256
16	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_5** Format: U4.8 Slope 5 for PWL for gamma correction The default is 512/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_4** Format: U4.8 Slope 4 for PWL for gamma correction The default is 658/256
17	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_7** Format: U4.8 Slope 7 for PWL for gamma correction The default is 278/256
	15:12	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE

	11:0	**PWL_Gamma_ Slope_6** Format: U4.8 Slope 6 for PWL for gamma correction The default is 368/256
18	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_9** Format: U4.8 Slope 9 for PWL for gamma correction The default is 179/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_8** Format: U4.8 Slope 8 for PWL for gamma correction The default is 215/256
19	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_11** Format: U4.8 Slope 11 for PWL for gamma correction The default is 124/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_10** Format: U4.8 Slope 10 for PWL for gamma correction The default is 151/256
20	31:24	**PWL_INV_GAMMA_ Point 4** Default Value: 101 Format: U8 Point 4 for PWL for inverse gamma correction

VEBOX_GAMUT_STATE

	23:16	**PWL_INV_GAMMA_ Point 3** Default Value: 79 Format: U8 Point 3 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Point 2** Default Value: 55 Format: U8 Point 2 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Point 1** Default Value: 30 Format: U8 Point 1 for PWL for inverse gamma correction
21	31:24	**PWL_INV_GAMMA_ Point 8** Format: U8 Point 8 for PWL for inverse gamma correction Value: 181 — Name:
	23:16	**PWL_INV_GAMMA_ Point 7** Format: U8 Point 7 for PWL for inverse gamma correction Value: 162 — Name:
	15:8	**PWL_INV_GAMMA_ Point 6** Format: U8 Point 6 for PWL for inverse gamma correction Value: 141 — Name:
	7:0	**PWL_INV_GAMMA_ Point 5** Format: U8 Point 5 for PWL for inverse gamma correction Value: 122 — Name:
22	31:24	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE

	23:16	**PWL_INV_GAMMA_ Point 11** Default Value: 237 Format: U8 Point 11 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Point 10** Default Value: 219 Format: U8 Point 10 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Point 9** Default Value: 200 Format: U8 Point 9 for PWL for inverse gamma correction
23	31:24	**PWL_INV_GAMMA_ Bias_4** Default Value: 33 Format: U8 Bias 4 for PWL for inverse gamma correction
	23:16	**PWL_INV_GAMMA_ Bias_3** Default Value: 20 Format: U8 Bias 3 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_2** Default Value: 10 Format: U8 Bias 2 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Bias_1** Default Value: 3 Format: U8 Bias 1 for PWL for inverse gamma correction
24	31:24	**PWL_INV_GAMMA_ Bias_8** Default Value: 117 Format: U8 Bias 8 for PWL for inverse gamma correction

VEBOX_GAMUT_STATE

	23:16	**PWL_INV_GAMMA_ Bias_7** Default Value: 92 Format: U8 Bias 7 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_6** Default Value: 67 Format: U8 Bias 6 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Bias_5** Default Value: 49 Format: U8 Bias 5 for PWL for inverse gamma correction
25	31:24	**Reserved** Format: MBZ
	23:16	**PWL_INV_GAMMA_ Bias_11** Default Value: 215 Format: U8 Bias 11 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_10** Default Value: 180 Format: U8 Bias 10 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Bias_9** Default Value: 147 Format: U8 Bias 9 for PWL for inverse gamma correction
26	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_1** Format: U4.8 Slope 1 for PWL for gamma correction The default is 72/256

VEBOX_GAMUT_STATE		
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_0** Format: U4.8 Slope 0 for PWL for gamma correction The default is 26/256
27	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_3** Format: U4.8 Slope 3 for PWL for gamma correction The default is 151/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_2** Format: U4.8 Slope 2 for PWL for gamma correction The default is 107/256
28	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_5** Format: U4.8 Slope 5 for PWL for gamma correction The default is 243/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_4** Format: U4.8 Slope 4 for PWL for gamma correction The default is 195/256
29	31:28	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE		
	27:16	**PWL_INV_GAMMA_ Slope_7** Format: U4.8 Slope 7 for PWL for gamma correction The default is 337/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_6** Format: U4.8 Slope 6 for PWL for gamma correction The default is 305/256
30	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_9** Format: U4.8 Slope 9 for PWL for gamma correction The default is 445/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_8** Format: U4.8 Slope 8 for PWL for gamma correction The default is 404/256
31	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_11** Format: U4.8 Slope 11 for PWL for gamma correction The default is 555/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_10** Format: U4.8

VEBOX_GAMUT_STATE

		Slope 10 for PWL for gamma correction The default is 498/256
32	31	**Reserved** Format: MBZ
	30:16	**Offset_in_G** Default Value: 0 Format: S14 The input offset for green component
	15	**Reserved** Format: MBZ
	14:0	**Offset_in_R** Default Value: 0 Format: S14 The input offset for red component
33	31	**Reserved** Format: MBZ
	30:16	**Offset_out_B** Format: S2.12 The input offset for green component The default is -1246/4096
	15	**Reserved** Format: MBZ
	14:0	**Offset_in_B** Default Value: 0 Format: S14 The input offset for red component
34	31	**Reserved** Format: MBZ
	30:16	**Offset_out_G** Format: S2.12 The input offset for green component The default is -983/4096

VEBOX_GAMUT_STATE

	15	**Reserved** Format: MBZ
	14:0	**Offset_out_R** Format: S2.12 The input offset for red component The default is -974/4096
35	31	**Reserved** Format: MBZ
	30	**FullRangeMappingEnable** 0 – Basic Mode **[Default]** 1 – Advance Mode
	29:20	**d(in,default)** Default Value: 205 Format: U10 InnerTriangleMappingLength
	19:10	**d(out, default)** Default Value: 164 Format: U10 OuterTriangleMappingLength
	9:0	**d1(out)** Default Value: 287 Format: U10 OuterTriangleMappingLengthBelow
36	31	**xvYccDecEncEnable** This bit is valid only when ColorGamutCompressionnEnable is on. 1 – Both xvYcc decode and xvYcc encode are enabled **[Default]** 0 – To disable both xvYcc decode and xvYcc encode
	30:28	**CompressionLineShift** 3 – **[Default]** [0,4]

VEBOX_GAMUT_STATE

	27:10	**Reserved** Format: MBZ
	9:0	**d1(in)** Default Value: 820 Format: U10 InnerTriangleMappingLengthBelow
37	31:30	**GCC BasicModeSelection** **Value** / **Name** / **Description** 00b / Default / 01b / Scaling Factor / Used along with Dword66 Bits 28:11 10b / Single Axis Gamma Correction / Used along with Dword67 Bit 29 11b / Scaling factor with fixed luma / Used along with Dword37 Bits 28:11
	29	**LumaChormaOnlyCorrection** **Value** / **Name** 0 / Luma Only Correction **[Default]** 1 / Chorma Only Correction
	28:25	**Reserved** Format: MBZ
	24:11	**BasicModeScalingFactor** Format: U2.12 Used when FullRangeMappingEnable is in basic mode and base mode selection bit is set to scaling factor.
	10:1	**Reserved** Format: MBZ
	0	**Cpi Override** **Value** / **Name** 0 / **[Default]** 1 / Override Cpi calculation

VEBOX_IECP_STATE

VEBOX_IECP_STATE		
Source:	VideoEnhancementCS	
Size (in bits):	2912	
Default Value:	0x9A6E39F0, 0x00000000, 0x00000000, 0x00000000

VEBOX_IECP_STATE		
67..69	95:0	**Alpha/AOI State** Format: **VEBOX_ALPHA_AOI_STATE** For description of this state, refer to *Alpha State Section*.
70..78	287:0	**CCM State** Format: **VEBOX_CCM_STATE** For description of this state, refer to the CCM State section
79..90	383:0	**Front-end CSC** Format: **VEBOX_FRONT_END_CSC_STATE** For description of this state, refer to *Front-end CSC State Section*.

VEBOX_PROCAMP_STATE

VEBOX_PROCAMP_STATE

Source: VideoEnhancementCS
Size (in bits): 64
Default Value: 0x01000001, 0x01000000

This state structure contains the IECP State Table Contents for ProcAmp state.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:17	**Contrast** Default Value: 80h = 1.0 in fixed point U4.7 Format: U4.7 Contrast magnitude.
	16:13	**Reserved** Format: MBZ
	12:1	**Brightness** Default Value: 0 or 0.0 Format: S7.4 2's complement Brightness magnitude.
	0	**PROCAMP Enable** Default Value: 1 Format: Enable
1	31:16	**Cos_c_s** Default Value: 256 Format: S7.8 2's complement UV multiplication cosine factor.
	15:0	**Sin_c_s** Default Value: 0 Format: S7.8 2's complement UV multiplication sine factor.

VEBOX_RGB_TO_GAMMA_CORRECTION

VEBOX_RGB_TO_GAMMA_CORRECTION		
Source:	VideoEnhancementCS	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Color depth is 16 bits.		
DWord	**Bit**	**Description**
0..1	63:48	**B-ch Corrected Value** Default Value: 0h Format: U16
	47:32	**G-ch Corrected Value** Default Value: 0h Format: U16
	31:16	**R-ch Corrected Value** Default Value: 0h Format: U16
	15:0	**Pixel Value** Default Value: 0h Format: U16 **Programming Notes** N indicates the index into the table. Pixel value 0 and Pixel Value 63 should be always programmed to 0 and 0xFFFF respectively. Rest of the Pixel Values which are used for comparing with the input pixel value should be multiple of 4.

VEBOX_STD_STE_STATE

VEBOX_STD_STE_STATE

Source: VideoEnhancementCS

Size (in bits): 928

Default Value: 0x9A6E39F0, 0x400D3C65, 0x000C9180, 0xFE2F2E00, 0x0003FFFF, 0x00140000, 0xD82E0640, 0x8285ECEC, 0x07FB8282, 0x00000000, 0x02117000, 0xA38FEC96, 0x0100C8C8, 0x003A6871, 0x01478000, 0x0107C306, 0x1291F008, 0x00094855, 0x1C1BD100, 0x03802008, 0x0002A980, 0x00080180, 0x0007CFF5, 0x18D1F07C, 0x000800BD, 0x1C080100, 0x03800000, 0x0008012B, 0x0008012B

This state structure contains the state used by the STD/STE function.

DWord	Bit	Description
0	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate.
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate.
	15:10	**Hue_Max** Default Value: 14 Format: U6 Rectangle half width.
	9:4	**Sat_Max** Default Value: 31 Format: U6 Rectangle half length.
	3	**Reserved** Format: MBZ
	2	**Output Control** Value 0: Output Pixels Value 1: Output STD Decisions

VEBOX_STD_STE_STATE

	1	**STE Enable** Format: Enable
	0	**STD Enable** Format: Enable **Programming Notes** This needs to be enabled if 'STD Score Output' is enabled.
1	31	**STD Score Output** Format: Enable
	30:28	**Diamond Margin** Default Value: 4 Format: U3
	27:21	**Diamond_du** Default Value: 0 Format: S6 2's complement Rhombus center shift in the sat-direction, relative to the rectangle center.
	20:18	**HS_margin** Default Value: 3 Format: U3 Defines rectangle margin.
	17:10	**Cos(α)** Default Value: 79 Format: S0.7 2's complement The default is 79/128
	9:8	**Reserved** Format: MBZ
	7:0	**Sin(α)** Default Value: 101 Format: S0.7 2's complement The default is 101/128
2	31:21	**Reserved** Format: MBZ

VEBOX_STD_STE_STATE

	20:13	**Diamond_alpha** Default Value: 100 Format: U2.6 1/tan(β) The default is 100/64
	12:7	**Diamond_Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.
	6:0	**Diamond_dv** Default Value: 0 Format: S6 2's complement Rhombus center shift in the hue-direction, relative to the rectangle center.
3	31:24	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	23:16	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	15:8	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
	7	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	6:0	**Reserved** Format: MBZ
4	31:18	**Reserved** Format: MBZ

VEBOX_STD_STE_STATE

	17:13	**Y_Slope_2** Default Value: 31 Format: U2.3 Slope between points Y3 and Y4. The default is 31/8
	12:8	**Y_Slope_1** Default Value: 31 Format: U2.3 Slope between points Y1 and Y2. The default is 31/8
	7:0	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function.
5	31:16	**INV_Skin_types_margin** Default Value: 20 Skin_Type_margin Format: U0.16 1/(2* Skin_types_margin)
	15:0	**INV_Margin_VYL** Format: U0.16 1 / Margin_VYL 1/ Margin_VYL = 3300/65536
6	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.

VEBOX_STD_STE_STATE

	15:0	**INV_Margin_VYU** Default Value: 1600 Format: U0.16 1 / Margin_VYU = 1600/65536
7	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y Point 2 of the lower part of the detection PWLF.
8	31:27	**Reserved** Format: MBZ
	26:16	**S0L** Default Value: FFBh Format: S2.8 2's complement Slope 0 of the lower part of the detection PWLF. The default is -5/256
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.

VEBOX_STD_STE_STATE

	7:0	**B2L** Default Value: 130 Format: U8 V Bias 2 of the lower part of the detection PWLF.
9	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Default Value: 0 Format: S2.8 2's complement The default is 0/256
	10:0	**S1L** Default Value: 0 Format: S2.8 2's complement Slope 1 of the lower part of the detection PWLF. The default is 0/256
10	31:27	**Reserved** Format: MBZ
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Default Value: 0 Format: S2.8 2's complement Slope 3 of the lower part of the detection PWLF. The default is 0/256

VEBOX_STD_STE_STATE

11	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
12	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Default Value: 256 Format: S2.8 2's complement Slope 0 of the upper part of the detection PWLF. The default is 256/256
	15:8	**B3U** Default Value: 200 Format: U8 V Bias 3 of the upper part of the detection PWLF.
	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.

VEBOX_STD_STE_STATE

13	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Default Value: F4Dh Format: S2.8 2's complement Slope 2 of the upper part of the detection PWLF. The default is -179/256
	10:0	**S1U** Default Value: 113 Format: S2.8 Slope 1 of the upper part of the detection PWLF. The default is 113/256
14	31:28	**Reserved** Format: MBZ
	27:20	**Skin_types_margin** Default Value: 20 Format: U8 Skin types Y margin Restrict Skin_types_thresh >= Skin_types_margin > 0 Restrict (Skin_types_thresh + Skin_types_margin) <= 255
	19:12	**Skin_types_thresh** Default Value: 120 Format: U8 Skin types Y margin Restrict Skin_types_thresh >= Skin_types_margin > 0 Restrict (Skin_types_thresh + Skin_types_margin) <= 255
	11	**Skin_Types_Enable** Default Value: 0 Disable Format: Enable Treat differently bright and dark skin types
	10:0	**S3U** Default Value: 0 Format: S2.8 2's complement

VEBOX_STD_STE_STATE

		Slope 3 of the upper part of the detection PWLF. The default is 0/256
15	31	**Reserved** Format: MBZ
	30:21	**SATB1** Default Value: 8 Format: S7.2 2's complement First bias for the saturation PWLF (bright skin). The default is 8/4
	20:14	**SATP3** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (bright skin).
	13:7	**SATP2** Default Value: 6 Format: S6 2's complement Second point for the saturation PWLF (bright skin).
	6:0	**SATP1** Default Value: 6 Format: S6 2's complement First point for the saturation PWLF (bright skin).
16	31	**Reserved** Format: MBZ
	30:20	**SATS0** Default Value: 297 Format: U3.8 Zeroth slope for the saturation PWLF (bright skin) The default is 297/256
	19:10	**SATB3** Default Value: 124 Format: S7.2 2's complement

VEBOX_STD_STE_STATE

		Third bias for the saturation PWLF (bright skin) The default is 124/4
	9:0	**SATB2** Default Value: 8 Format: S7.2 2's complement Second bias for the saturation PWLF (bright skin) The default is 8/4
17	31:22	**Reserved** Format: MBZ
	21:11	**SATS2** Default Value: 297 Format: U3.8 Second slope for the saturation PWLF (bright skin) The default is 297/256
	10:0	**SATS1** Default Value: 85 Format: U3.8 First slope for the saturation PWLF (bright skin) The default is 85/256
18	31:25	**HUEP3** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (bright skin)
	24:18	**HUEP2** Default Value: 6 Format: S6 2's complement Second point for the hue PWLF (bright skin)
	17:11	**HUEP1** Default Value: 7Ah -6 Format: S6 2's complement First point for the hue PWLF (bright skin)

VEBOX_STD_STE_STATE

	10:0	**SATS3** Default Value: 256 Format: U3.8 Third slope for the saturation PWLF (bright skin) The default is 256/256
19	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3** Default Value: 56 Format: S7.2 2's complement Third bias for the hue PWLF (bright skin) The default is 56/4
	19:10	**HUEB2** Default Value: 8 Format: S7.2 2's complement Second bias for the hue PWLF (bright skin) The default is 8/4
	9:0	**HUEB1** Default Value: 8 Format: S7.2 2's complement First bias for the hue PWLF (bright skin) The default is 8/4
20	31:22	**Reserved** Format: MBZ
	21:11	**HUES1** Default Value: 85 Format: U3.8 First slope for the hue PWLF (bright skin) The default is 85/256
	10:0	**HUES0** Default Value: 384 Format: U3.8

VEBOX_STD_STE_STATE

		Zeroth slope for the hue PWLF (bright skin) The default is 384/256
21	31:22	**Reserved** Format: MBZ
	21:11	**HUES3** Default Value: 256 Format: U3.8 Third slope for the hue PWLF (bright skin) The default is 256/256
	10:0	**HUES2** Default Value: 384 Format: U3.8 Second slope for the hue PWLF (bright skin) The default is 384/256
22	31	**Reserved** Format: MBZ
	30:21	**SATB1_DARK** Default Value: 0 Format: S7.2 2's complement First bias for the saturation PWLF (dark skin) The default is 0/4
	20:14	**SATP3_DARK** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (dark skin)
	13:7	**SATP2_DARK** Default Value: 31 Format: S6 2's complement Second point for the saturation PWLF (dark skin)

VEBOX_STD_STE_STATE

	6:0	**SATP1_DARK** Default Value: FF5h Format: S6 2's complement First point for the saturation PWLF (dark skin) Default Value: -11
23	31	**Reserved** Format: MBZ
	30:20	**SATS0_DARK** Default Value: 397 Format: U3.8 Zeroth slope for the saturation PWLF (dark skin) The default is 397/256
	19:10	**SATB3_DARK** Default Value: 124 Format: S7.2 2's complement Third bias for the saturation PWLF (dark skin) The default is 124/4
	9:0	**SATB2_DARK** Default Value: 124 Format: S7.2 2's complement Second bias for the saturation PWLF (dark skin) The default is 124/4
24	31:22	**Reserved** Format: MBZ
	21:11	**SATS2_DARK** Default Value: 256 Format: U3.8 Second slope for the saturation PWLF (dark skin) The default is 256/256
	10:0	**SATS1_DARK** Default Value: 189 Format: U3.8

VEBOX_STD_STE_STATE

		First slope for the saturation PWLF (dark skin) The default is 189/256
25	31:25	**HUEP3_DARK** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	24:18	**HUEP2_DARK** Default Value: 2 Format: S6 2's complement Second point for the hue PWLF (dark skin).
	17:11	**HUEP1_DARK** Default Value: 0 Format: S6 2's complement First point for the hue PWLF (dark skin).
	10:0	**SATS3_DARK** Default Value: 256 Format: U3.8 Third slope for the saturation PWLF (dark skin) The default is 256/256
26	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3_DARK** Default Value: 56 Format: S7.2 2's complement Third bias for the hue PWLF (dark skin). The default is 56/4
	19:10	**HUEB2_DARK** Default Value: 0 Format: S7.2 2's complement Second bias for the hue PWLF (dark skin). The default is 0/4

VEBOX_STD_STE_STATE

	9:0	**HUEB1_DARK** Default Value: 0 Format: S7.2 2's complement First bias for the hue PWLF (dark skin). The default is 0/4
27	31:22	**Reserved** Format: MBZ
	21:11	**HUES1_DARK** Default Value: 256 Format: U3.8 First slope for the hue PWLF (dark skin). The default is 256/256
	10:0	**HUES0_DARK** Default Value: 299 Format: U3.8 Zeroth slope for the hue PWLF (dark skin). The default is 299/256
28	31:22	**Reserved** Format: MBZ
	21:11	**HUES3_DARK** Default Value: 256 Format: U3.8 Third slope for the hue PWLF (dark skin). The default is 256/256
	10:0	**HUES2_DARK** Default Value: 299 Format: U3.8 Second slope for the hue PWLF (dark skin). The default is 299/256

VEBOX_TCC_STATE

VEBOX_TCC_STATE	
Source:	VideoEnhancementCS
Size (in bits):	352
Default Value:	0xDCDCDC00, 0xDCDCDC00, 0x1E34CC91, 0x3E3CCE91, 0x02E80195, 0x0197046B, 0x01790174, 0x00096000, 0x00000000, 0x03030000, 0x009201C0

This state structure contains the IECP State Table Contents for TCC state.

DWord	Bit	Description
0	31:24	**SatFactor3** Default Value: 220 Format: U1.7 The saturation factor for yellow. The default is 220/128
	23:16	**SatFactor2** Default Value: 220 Format: U1.7 The saturation factor for red. The default is 220/128
	15:8	**SatFactor1** Default Value: 220 Format: U1.7 The saturation factor for magenta. The default is 220/128
	7	**TCC Enable** Format: Enable
	6:0	**Reserved** Format: MBZ
1	31:24	**SatFactor6** Default Value: 220 Format: U1.7 The saturation factor for blue. The default is 220/128

VEBOX_TCC_STATE

	23:16	**SatFactor5** Default Value: 220 Format: U1.7 The saturation factor for cyan. The default is 220/128
	15:8	**SatFactor4** Default Value: 220 Format: U1.7 The saturation factor for green. The default is 220/128
	7:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ
	29:20	**BaseColor3** Default Value: 483 Format: U10 Base Color 3 - this value must be greater than BaseColor2
	19:10	**BaseColor2** Default Value: 307 Format: U10 Base Color 2 - this value must be greater than BaseColor1
	9:0	**BaseColor1** Default Value: 145 Format: U10 Base Color 1
3	31:30	**Reserved** Format: MBZ
	29:20	**BaseColor6** Default Value: 995 Format: U10 Base Color 6 - this value must be greater than BaseColor5

VEBOX_TCC_STATE

	19:10	**BaseColor5** Default Value: 819 Format: U10 Base Color 5 - this value must be greater than BaseColor4
	9:0	**BaseColo4** Default Value: 657 Format: U10 Base Color 4 - this value must be greater than BaseColor3
4	31:16	**ColorTransitSlope23** Default Value: 744 Format: U0.16 The calculation result of 1 / (BC3 - BC2) [1/62]
	15:0	**ColorTransitSlope2** Default Value: 405 Format: U0.16 The calculation result of 1 / (BC2 - BC1) [1/57]
5	31:16	**ColorTransitSlope45** Default Value: 407 Format: U0.16 The calculation result of 1 / (BC5 - BC4) [1/57]
	15:0	**ColorTransitSlope34** Default Value: 1131 Format: U0.16 The calculation result of 1 / (BC4 - BC3) [1/61]
6	31:16	**ColorTransitSlope61** Default Value: 377 Format: U0.16 The calculation result of 1 / (BC1 - BC6) [1/62]
	15:0	**ColorTransitSlope56** Default Value: 372 Format: U0.16 The calculation result of 1 / (BC6 - BC5) [1/62]

VEBOX_TCC_STATE

7	31:22	**ColorBias3** Default Value: 0 Format: U2.8 Color bias for BaseColor3.
	21:12	**ColorBias2** Default Value: 150 Format: U2.8 Color bias for BaseColor2. The default is 150/256
	11:2	**ColorBias1** Default Value: 0 Format: U2.8 Color bias for BaseColor1.
	1:0	**Reserved** Format: MBZ
8	31:22	**ColorBias6** Default Value: 0 Format: U2.8 Color bias for BaseColor6.
	21:12	**ColorBias5** Default Value: 0 Format: U2.8 Color bias for BaseColor5.
	11:2	**ColorBias4** Default Value: 0 Format: U2.8 Color bias for BaseColor4.
	1:0	**Reserved** Format: MBZ
9	31	**Reserved** Format: MBZ

VEBOX_TCC_STATE

	30:24	**UV Threshold** Default Value: 3 Format: U7 Low UV threshold.
	23:19	**Reserved** Format: MBZ
	18:16	**UV Threshold Bits** Default Value: 3 Format: U3 Low UV transition width bits.
	15:13	**Reserved** Format: MBZ
	12:8	**STE Threshold** Default Value: 0 Format: U5 Skin tone pixels enhancement threshold.
	7:3	**Reserved** Format: MBZ
	2:0	**STE Slope Bits** Default Value: 0 Format: U3 Skin tone pixels enhancement slope bits.
10	31:16	**Inv_UVMaxColor** Default Value: 146 Format: U16 1 / UVMaxColor. Used for the SFs2 calculation.
	15:9	**Reserved** Format: MBZ
	8:0	**UVMaxColor** Default Value: 448 Format: U9 The maximum absolute value of the legal UV pixels. Used for the SFs2 calculation.

VEBOX_VERTEX_TABLE

VEBOX_VERTEX_TABLE	
Source:	VideoEnhancementCS
Size (in bits):	16384
Default Value:	0x00000000, 0x00000000, 0x00000000,

VEBOX_VERTEX_TABLE

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

VEBOX_VERTEX_TABLE

0x00000000, 0x00000000

DWord	Bit	Description
0..511	31:28	**Reserved** Format: MBZ
	27:16	**Vertex table entry 0 - Lv (12 bits)** Value: 100h-ED6h; Name: ; Description: Range for Vertices BT601 and BT709
	15:12	**Reserved** Format: MBZ
	11:0	**Vertex table entry 0 - Cv (12 bits)** Value: 400h-A00h; Name: ; Description: Range for Vertices BT601 and BT709

VECS Hardware-Detected Error Bit Definitions

VECS Hardware-Detected Error Bit Definitions

Source: VideoEnhancementCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: **Value** / **Name** / **Description**: 1 / / Instruction Error detected **Programming Notes**: This error indications cannot be cleared except by reset (i.e., it is a fatal error).

VERTEX_BUFFER_STATE

VERTEX_BUFFER_STATE

Source: RenderCS
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure is used in 3DSTATE_VERTEX_BUFFERS to set the state associated with a VB. The VF function will use this state to determine how/where to extract vertex element data for all vertex elements associated with the VB.

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Format: U6 index This field contains an index value which selects the VB state being defined. Value: [0,32] — Name:
	25:23	**Reserved** Format: MBZ
	22:16	**Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this vertex buffer.
	15	**Reserved** Format: MBZ
	14	**Address Modify Enable** If set, the Buffer Starting Address field is used to update the state of this buffer. If clear, that field is ignored and the previously-programmed value is maintained.
	13	**Null Vertex Buffer** Format: Enable This field enabled causes any fetch for vertex data to return 0. Programming Notes: VERTEX_BUFFER_STATE.Null Vertex Buffer must be set when the VERTEX_BUFFER_STATE.Buffer Size is 0x0.
	12	**Reserved** Format: MBZ
	11:0	**Buffer Pitch** Format: U12 Count of bytes This field specifies the pitch in bytes of the structures accessed within the VB. This information is required in order to access elements in the VB via a structure index.

VERTEX_BUFFER_STATE

Value	Name	Description
[0,4095]		Bytes

Programming Notes

- Different VERTEX_BUFFER_STATE structures can refer to the same memory region using different Buffer Pitch values.
- See note on 64-bit float alignment in Buffer Starting Address.

1..2 | 63:0 | **Buffer Starting Address**

Format:	GraphicsAddress[63:0]Vertex_Buffer

This field contains the byte-aligned Graphics Address LSBs of the first element of interest within the VB. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer.

If the Address ModifyEnable bit is clear, this field is ignored and the previous value of Buffer Starting Address for this buffer is maintained.

Programming Notes

- 64-bit floating point values must be 64-bit aligned in memory, or UNPREDICTABLE data will be fetched. When accessing an element containing 64-bit floating point values, the Buffer Starting Address and Source Element Offset values must add to a 64-bit aligned address, and BufferPitch must be a multiple of 64-bits.
- VBs can only be allocated in linear (not tiled) graphics memory.
- As computed index values are, by definition, interpreted as unsigned values, there is no issue with accesses to locations before (lower address value) the start of the buffer. However, these wrapped indices are subject to Max Index checking (see below).

3 | 31:0 | **Buffer Size**

Format:	U32 Count of bytes

This field specifies the size of the buffer in bytes. Vertex element accesses which straddle or go past the end of the buffer will return 0's for all elements.Note that BufferSize=0 indicates that there is no valid data in the buffer.

Value	Name
[0, FFFFFFFFh]	

VERTEX_ELEMENT_STATE

VERTEX_ELEMENT_STATE

Source:	RenderCS
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

Description

This structure is used in 3DSTATE_VERTEX_ELEMENTS to set the state associated with a vertex element. A vertex element is defined as an entity supplying from one to four DWord vertex components, to be stored in the vertex URB entry.

The number of supported vertex elements is 34.

The VF function will use this state, and possibly the state of the associated vertex buffer, to fetch/generate the source vertex element data, perform any required format conversions, padding with zeros, and store the resulting destination vertex element data into the vertex URB entry.

Programming Notes

- The (new) 3DSTATE_VF_SGVS command is used to specify optional insertion of VertexID and/or InstanceID into the input vertex data, logically following the processing of the VERTEX_ELEMENT_STATE structures. The VFCOMP_STORE_VID/IID encodings are no longer available in VERTEX_ELEMENT_STATE.
- When SourceElementFormat is set to one of the *64*_PASSTHRU formats, 64-bit components are stored in the URB without any conversion. In this case, vertex elements must be written as 128 or 256 bits, with VFCOMP_STORE_0 being used to pad the output as required. E.g., if R64_PASSTHRU is used to copy a 64-bit Red component into the URB, Component 1 must be specified as VFCOMP_STORE_0 (with Components 2,3 set to VFCOMP_NOSTORE) in order to output a 128-bit vertex element, or Components 1-3 must be specified as VFCOMP_STORE_0 in order to output a 256-bit vertex element. Likewise, use of R64G64B64_PASSTHRU requires Component 3 to be specified as VFCOMP_STORE_0 in order to output a 256-bit vertex element.
- When SourceElementFormat is set to one of the *64*_PASSTHRU formats then VFCOMP_STORE_SRC must be used for every valid component.
- Any SourceElementFormat of *64*_PASSTHRU cannot be used with an element which has edge flag enabled.

The SourceElementFormat needs to be a single-component format with an element which has edge flag enabled.

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Format: U6 This field specifies which vertex buffer the element is sourced from. [0,32] — Up to 33 VBs are supported

VERTEX_ELEMENT_STATE

Programming Notes

It is possible for a vertex element to include only internally-generated data (VertexID, etc.), in which case the associated vertex buffer state is ignored.

25 **Valid**

Format:	Boolean

Value	Name	Description
1h	TRUE	this vertex element is used in vertex assembly
0h	FALSE	this vertex element is not used.

24:16 **Source Element Format**

Format:	SURFACE_FORMAT

Range: Valid formats are found in the 3D Primitive Processing FormatConversion portion of the vertex fetch chapter.

Format: The encoding of this field is identical the Surface Format field of the SURFACE_STATE structure, as described in the Sampler chapter.

This field specifies the format in which the memory-resident source data for this particular vertex element is stored in the memory buffer. This only applies to elements stored with VFCOMP_STORE_SRC component control. (All other component types have an explicit format).

15 **Edge Flag Enable**

Format:	Enable

Description

When ENABLED, the source element is interpreted as an EdgeFlag for the vertex. If the source element is zero, the EdgeFlag will be set to FALSE. If the source element is non-zero, the EdgeFlag will be set to TRUE. The EdgeFlag bit will travel down the fixed function pipeline along with the vertex handle, etc. and not be stored in the vertex data like the other vertex elements. Refer to the fixed function descriptions for how this EdgeFlag affects rendering.
Edge flags are supported for the following primitive topology types only, otherwise EdgeFlagEnable must not be ENABLED.

- 3DPRIM_TRILIST*
- 3DPRIM_TRISTRIP*
- 3DPRIM_TRIFAN*
- 3DPRIM_POLYGON

If this bit is DISABLED for all valid VERTEX_ELEMENTs, the vertex will be assigned a default EdgeFlag of TRUE.

Edge flags are supported for all primitive topology types.

Programming Notes

VERTEX_ELEMENT_STATE

- This bit must only be ENABLED on the last valid VERTEX_ELEMENT structure.
- When set, Component 0 Control must be set to VFCOMP_STORE_SRC, and Component 1-3 Control must be set to VFCOMP_NOSTORE.

	14:12	**Reserved** Format: MBZ
	11:0	**Source Element Offset** Format: U12 byte offset Byte offset of the source vertex element data in the structures comprising the vertex buffer. **Value**: [0,2047] **Name**: **Programming Notes**: See note on 64-bit float alignment in Buffer Starting Address.
1	31	**Reserved** Format: MBZ
	30:28	**Component 0 Control** Format: 3D_Vertex_Component_Control Refer to the 3D_Vertex_Component_Control table below
	27	**Reserved** Format: MBZ
	26:24	**Component 1 Control** Format: 3D_Vertex_Component_Control Refer to the 3D_Vertex_Component_Control table below
	23	**Reserved** Format: MBZ
	22:20	**Component 2 Control** Format: 3D_Vertex_Component_Control Refer to the 3D_Vertex_Component_Control table below
	19	**Reserved** Format: MBZ
	18:16	**Component 3 Control** Format: 3D_Vertex_Component_Control Refer to the 3D_Vertex_Component_Control table below

VERTEX_ELEMENT_STATE		
	15:8	**Reserved** Format: MBZ
	7:0	**Reserved** Format: MBZ

Vertical Line Stride Override Message Descriptor Control Field

MDC_VLSO - Vertical Line Stride Override Message Descriptor Control Field

Source:	BSpec
Size (in bits):	3
Default Value:	0x00000000

DWord	Bit	Description
0	2	**Vertical Line Stride Override** Format: Enable If set, override the Vertical Line Stride and Vertical Line Stride Offset fields in the surface state with the fields below.
	1	**Vertical Line Stride** Format: U1 Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.
	0	**Vertical Line Stride Offset** Format: U1 Specifies the offset of the initial line from the beginning of the buffer. Ignored when Override VerticalLine Stride is 0.

VFE_STATE_EX

VFE_STATE_EX		
Source:	RenderCS	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:8	**Reserved**
	7:0	**Reserved** Format: MBZ
1	31:0	**VFE Control** This field is used by VFE depending on the mode of operation. See the following tables for details. If VFE Mode = AVC-IT or AVC-MC, this field is valid as defined in Table 1 13. If VFE Mode = VC1-IT, this field is valid as defined in Table 1 14. Otherwise, this field is reserved.
2	31:0	**Interface Descriptor Remap Table** This field contains the interface descriptor remap table entries for the first 8 kernel indices. Each table entry has 4 bits, providing a remapping range of [0, 15]. The input of this table is the Interface Descriptor Offset within the MEDIA_OBJECT or MEDIA_OBJECT_EX command. As the table is limited to map the first 16 values, any Interface Descriptor Offset greater than 15 is not remapped. Bits 31:28: Remap for index = 7 Bits 27:24: Remap for index = 6 Bits 23:20: Remap for index = 5 Bits 19:16: Remap for index = 4 Bits 15:12: Remap for index = 3 Bits 11:8: Remap for index = 2 Bits 7:4: Remap for index = 1 Bits 3:0: Remap for index = 0
3	31:0	**Interface Descriptor Remap Table (cont)** This field contains the interface descriptor remap table entries for the next 8 kernel indices (index = 8...15). Each table entry has 4 bits, providing a remapping range of [0, 15]. Bits 31:28: Remap for index = 15 Bits 27:24: Remap for index = 14 Bits 23:20: Remap for index = 13 Bits 19:16: Remap for index = 12 Bits 15:12: Remap for index = 11 Bits 11:8: Remap for index = 10

VFE_STATE_EX

		Bits 7:4: Remap for index = 9 Bits 3:0: Remap for index = 8
4	31	**Scoreboard Enable** This field enables and disables the hardware scoreboard in the Media Pipeline. If this field is cleared, hardware ignores the following scoreboard state fields. This should be enabled at all times in the state and the scoreboard enable field in the MEDIA_OBJECT command should be use instead. If this field is disabled, the scratch space pointer calculation will be incorrect and any attempt to use the scoreboard later will result in a hardware hang. **Value** / **Name** 0 — Scoreboard disabled 1 — Scoreboard enabled
	30	**Scoreboard Type** This field selects the type of scoreboard in use. This field must be zero (stalling scoreboard) **Value** / **Name** 0 — Stalling Scoreboard 1 — Reserved (for Non-stalling scoreboard)
	29:8	**Reserved** Format: MBZ
	7:0	**Scoreboard Mask** Format: Boolean Each bit indicates the corresponding dependency scoreboard is enabled. The scoreboard is based on the relative (X, Y) distance from the current threads' (X, Y) position. **Value** / **Name** / **Description** [0,7] / Bit n / Score n is enabled
5	31:28	**Scoreboard 3 Delta Y** Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.
	27:24	**Scoreboard 3 Delta X** Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.

VFE_STATE_EX		
	23:16	**Scoreboard 2 Delta (X, Y)**
	15:8	**Scoreboard 1 Delta (X, Y)**
	7:0	**Scoreboard 0 Delta (X, Y)**
6	31:24	**Scoreboard 7 Delta (X, Y)**
	23:16	**Scoreboard 6 Delta (X, Y)**
	15:8	**Scoreboard 5 Delta (X, Y)**
	7:0	**Scoreboard 4 Delta (X, Y)**
7	31:0	**Reserved** Format: MBZ

VP8 Encoder StreamOut Format

VP8 Encoder StreamOut Format		
Source:	VideoCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:24	**MbY** Format: U8
	23:16	**MbX** Format: U8
	15:8	**MbClock16** Format: U8
	7:3	**Reserved** Format: MBZ
	2	**MbRcFlag** Format: U1
	1	**MBLevelInterMBConformanceFlag** Format: U1
	0	**MBLevelIntraMBConformanceFlag** Format: U1
1	31:29	**Reserved** Format: MBZ
	28:16	**MB_Residual_BitCount** Format: U13
	15:13	**Reserved** Format: MBZ
	12:0	**MB_Total_BitCount** Format: U13
2	31:25	**Reserved** Format: MBZ
	24:0	**Cbp** Format: U25
3	31	**Reserved** Format: MBZ

VP8 Encoder StreamOut Format			
	30	**LastMbFlag**	
		Format:	U1
	29	**IntraMBFlag**	
		Format:	U1
	28:24	**MBType5Bits**	
		Format:	U5
	23:19	**Reserved**	
		Format:	MBZ
	18	**QindexClampHigh**	
		Format:	U1
	17	**QindexClampLow**	
		Format:	U1
	16	**CoeffClampStatus**	
		Format:	U1
	15:0	**Reserved**	
		Format:	MBZ

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 3: GPU Overview

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Introduction **1**
Graphics Processing Unit (GPU) 1
GPU Overview 2
Command Stream (CS) Unit 3
3D Pipeline 3
Media Pipeline 3
Thread Dispatching 3
Execution Units (EUs) 3
Shared Functions 4
Fixed and Shared Function IDs 4
Register Address Maps **5**
Graphics Register Address Map 5
Memory and IO Space Registers 5
VGA and Extended VGA Register Map 5
VGA and Extended VGA I/O and Memory Register Map 6
Indirect VGA and Extended VGA Register Indices 7
Memory Object Overview **10**
Hardware Status Page 11
Memory Access Indirection 11
Instruction Ring Buffers 13
Instruction Batch Buffers 13
Logical Contexts 14
BSD Logical Render Context Address (LRCA) 14
Overall Context Layout [..... 14
Register/State Context 15
Ring Buffer 16
Ring Context 16
The Per-Process Hardware Status 17
Video Engine Power Context 17
Copy Engine Logical Context Data 20
Context Layout 20

Register/State Context....20
Ring Buffer....21
Ring Context....22
The Per-Process Hardware Status Page....22
Blitter Engine Power Context....23
Video Enhancement Logical Context Data....25
Context Layout....25
Ring Context....25
Register/State Context....26
Video Enhancement Engine Power Context....28
Memory Data Formats....30
Unsigned Normalized (UNORM)....30
Gamma Conversion (SRGB)....30
Signed Normalized (SNORM)....30
Unsigned Integer (UINT/USCALED)....30
Signed Integer (SINT/SSCALED)....31
Floating Point (FLOAT)....31
64-bit Floating Point....31
32-bit Floating Point....31
16-bit Floating Point....32
11-bit Floating Point....33
10-bit Floating Point....33
Shared Exponent....34
Memory Compression....34
Media Memory Compression....34

Introduction

The integrated graphics component, specifically called the Graphics Processing Unit, or GPU, resides on the same chip die as the Central Processing Unit, or CPU, and communicates with the CPU via the on-chip bus, with internal memory and with output device(s). As Intel GPUs have evolved, they now occupy a significant percentage of space on the chip, and provide customers with high performance and low-power graphics processing, eliminating the need to purchase a separate video card for most users.

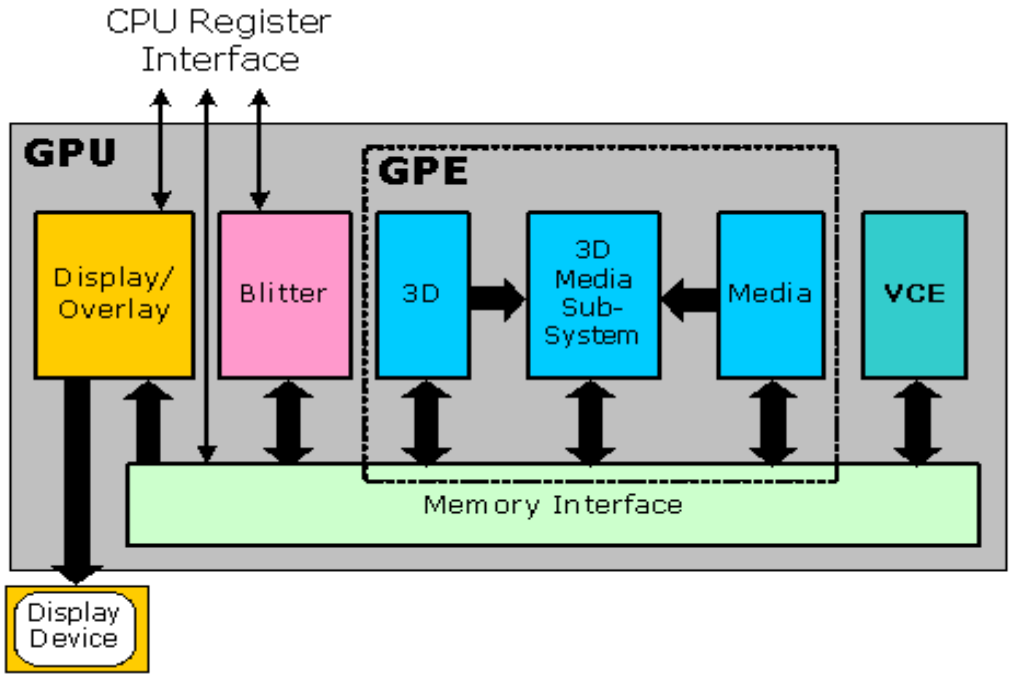
This Behavioral Specification, or BSpec for short, provides detailed narrative and referential information required by graphics device driver engineers and graphics API-level programmers to take advantage of the sophisticated architecture and programmability of the GPU.

Graphics Processing Unit (GPU)

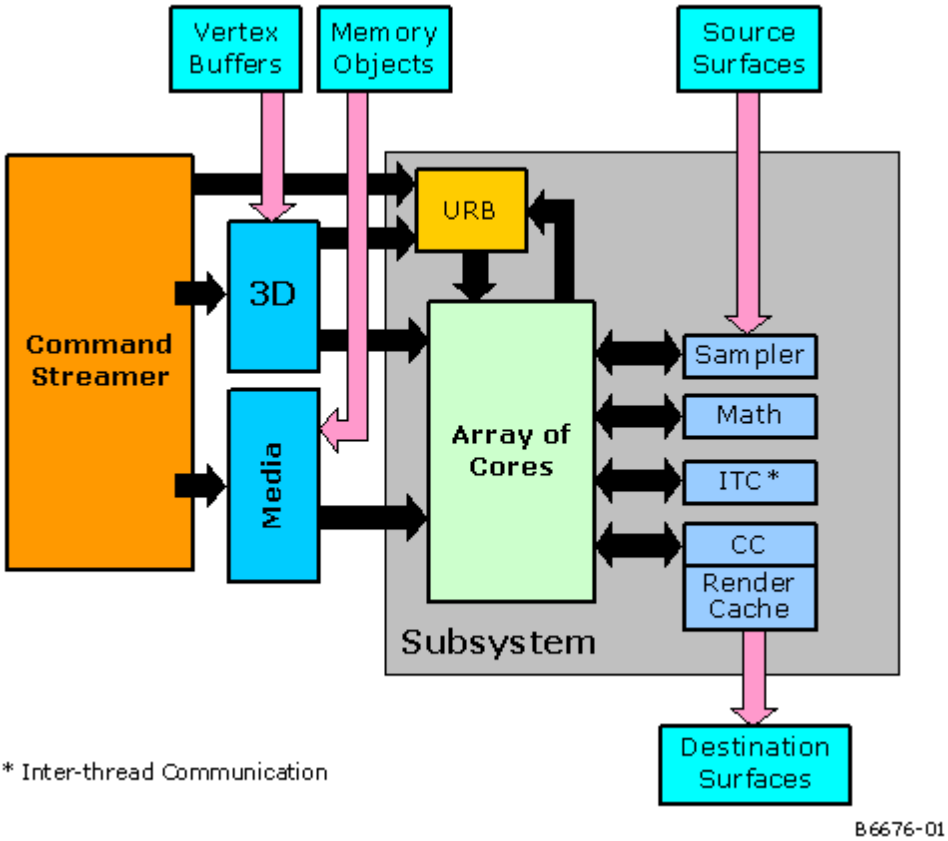
The Graphics Processing Unit is controlled by the CPU through a direct interface of memory-mapped IO registers, and indirectly by parsing commands that the CPU has placed in memory. The Display interface and Blitter (**bl**ock **i**mage **t**ransfer**er**) are controlled primarily by direct CPU register addresses, while the 3D and Media pipelines and the parallel Video Codec Engine (VCE) are controlled primarily through instruction lists in memory.

The subsystem contains an array of cores, or execution units, with a number of “shared functions”, which receive and process messages at the request of programs running on the cores. The shared functions perform critical tasks, such as sampling textures and updating the render target (usually the frame buffer). The cores themselves are described by an instruction set architecture, or ISA.

Block Diagram of the GPU



GPU Overview



The subsystem consists of an array of *execution units* (*EUs*, sometimes referred to as an array of *cores*) along with a set of *shared functions* outside the EUs that the EUs leverage for I/O and for complex computations. Programmers access the subsystem via the 3D or Media pipelines.

EUs are general-purpose programmable cores that support a rich instruction set that has been optimized to support various 3D API shader languages as well as media functions (primarily video) processing.

Shared functions are hardware units which serve to provide specialized supplemental functionality for the EUs. A shared function is implemented where the demand for a given specialized function is insufficient to justify the costs on a per-EU basis. Instead a single instantiation of that specialized function is implemented as a stand-alone entity outside the EUs and shared among the EUs.

Invocation of the shared functionality is performed via a communication mechanism called a *message*. A message is a small self-contained packet of information created by a kernel and directed to a specific shared function. The message is defined by a range of MRF registers that hold message operands, a destination shared function ID, a function-specific encoding of the desired operation, and a destination GRF register to which any writeback response is to be directed. Messages are dispatched to the shared function under software control via the send instruction. This instruction identifies the contents of the message and the GRF register locations to direct any response.

The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

Command Stream (CS) Unit

The Command Stream (CS) unit manages the use of the 3D and Media pipelines; it performs switching between pipelines and forwarding command streams to the currently active pipeline. It manages allocation of the URB and helps support the Constant URB Entry (CURBE) function.

3D Pipeline

The 3D Pipeline provides specialized 3D primitive processing functions. These functions are provided by a pipeline of "fixed function" stages (units) and GEN threads spawned by these units. See *3D Pipeline Overview*.

Media Pipeline

The Media pipeline provides both specialized media-related processing functions and the ability to perform more general ("generic") functionality. These Media-specific functions are provided by a Video Front End (VFE) unit. A Thread Spawner (TS) unit is utilized to spawn GEN threads requested by the VFE unit, or as required when the pipeline is used for general processing. See *Media Pipeline Overview*.

Thread Dispatching

When the 3D and Media pipelines send requests for thread initiation to the Subsystem, the thread Dispatcher receives the requests. The dispatcher performs such tasks as arbitrating between concurrent requests, assigning requested threads to hardware threads on EUs, allocating register space in each EU among multiple threads, and initializing a thread's registers with data from the fixed functions and from the URB. This operation is largely transparent to software.

Execution Units (EUs)

The Execution Units (EUs) are the programmable shader units of the Gen Architecture. Each is a stand-alone programmable computational unit used for execution of 3D shaders and media/gpgpu kernels. Internally each is capable of multi-issue SIMD execution, and their hardware multi-threaded operation provides a very high-efficiency execution environment in the face of long data latencies typically associated with memory accesses. Each hardware thread within an EU has a dedicated large-capacity high-bandwidth register file (GRF) and associated independent thread-state. Execution is multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other miscellaneous operations. Communication to support units (shared functions) for operations such as texture sampling or scatter/gather load/stores is via 'messages' programmatically constructed and 'sent' to those functions, with dependency hardware causing the issuing thread to sleep until the requested data has been returned.

EU instance count varies by product generation, as well as by SKU within a given generation, and their capabilities have evolved over the many generation of the Gen Architecture.. Please see "Device Attributes" in the "Configuration" chapter for specific rates and capacities associated with Execution Units.

Shared Functions

Shared functions are hardware units that provide specialized supplemental functionality for the EUs. A shared function is implemented where the demand for a given specialized function is insufficient to justify the costs on a per-EU basis. Instead a single instantiation of that specialized function is implemented as a stand-alone entity outside the EUs and shared among the EUs.

Invocation of the shared functionality is performed via a communication mechanism called a message. A message is a small self-contained packet of information created by a kernel and directed to a specific shared function.

Programming Note	
Context:	Communication mechanism in shared functions
The message is defined by a range of Message Register File (MRF) registers that hold message operands, a destination shared function ID, a function-specific encoding of the desired operation, and a destination General Register File (GRF) register to which any writeback response is directed.	

Messages are dispatched to the shared function under software control via the *send* instruction. This instruction identifies the contents of the message and the GRF register locations to direct any response.

The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

Fixed and Shared Function IDs

The following lists the assignments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a *send* instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function.

Function IDs

[Enumeration] Fixed Function ID
[Enumeration] Shared Function ID

Register Address Maps

Graphics Register Address Map

This chapter provides address maps of the graphics controllers I/O and memory-mapped registers. Individual register bit field descriptions are provided in the following chapters. PCI configuration address maps and register bit descriptions are provided in the following chapter.

Memory and IO Space Registers

These are graphics MMIO ranges. Note that this is only a subset of the complete definition of the MMIO address space.

Range Start (Hex)	Range End (Hex)	Unit Owning the Range
00002000	00002FFF	Render/Generic Media Engine
00004000	00004FFF	Render/Generic Media Graphics Memory Arbiter
00012000	000123FF	MFX Control Engine (Video Command Streamer)
00012400	00012FFF	Media Units (VIN Unit)
00014000	00014FFF	MFX Memory Arbiter
00022000	00022FFF	Blitter Engine
00024000	00024FFF	Blitter Memory Arbiter
00100000	00107FFF	Fence Registers
00140000	0017FFFF	MCHBAR (SA)

Programming Note	
Context:	Memory and IO space registers, graphics MMIO ranges
8800h-88FFh is a reserved range for GT. IA accesses to this region have no impact.	

VGA and Extended VGA Register Map

For I/O locations, the value in the address column represents the register I/O address. For memory mapped locations, this address is an offset from the base address programmed in the MMADR register.

VGA and Extended VGA I/O and Memory Register Map

Address	Register Name (Read)	Register Name (Write)
2D Registers		
3B0h–3B3h	Reserved	Reserved
3B4h	VGA CRTC Index (CRX) (monochrome)	VGA CRTC Index (CRX) (monochrome)
3B5h	VGA CRTC Data (monochrome)	VGA CRTC Data (monochrome)
3B6h–3B9h	Reserved	Reserved
3Bah	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3BBh–3BFh	Reserved	Reserved
3C0h	VGA Attribute Controller Index (ARX)	VGA Attribute Controller Index (ARX)/ VGA Attribute Controller Data (alternating writes select ARX or write ARxx Data)
3C1h	VGA Attribute Controller Data (read ARxx data)	Reserved
3C2h	VGA Feature Read Register (ST00)	VGA Miscellaneous Output Register (MSR)
3C3h	Reserved	Reserved
3C4h	VGA Sequencer Index (SRX)	VGA Sequencer Index (SRX)
3C5h	VGA Sequencer Data (SRxx)	VGA Sequencer Data (SRxx)
3C6h	VGA Color Palette Mask (DACMASK)	VGA Color Palette Mask (DACMASK)
3C7h	VGA Color Palette State (DACSTATE)	VGA Color Palette Read Mode Index (DACRX)
3C8h	VGA Color Palette Write Mode Index (DACWX)	VGA Color Palette Write Mode Index (DACWX)
3C9h	VGA Color Palette Data (DACDATA)	VGA Color Palette Data (DACDATA)
3CAh	VGA Feature Control Register (FCR)	Reserved
3CBh	Reserved	Reserved
3CCh	VGA Miscellaneous Output Register (MSR)	Reserved
3CDh	Reserved	Reserved
3CEh	VGA Graphics Controller Index (GRX)	VGA Graphics Controller Index (GRX)
3CFh	VGA Graphics Controller Data (GRxx)	VGA Graphics Controller Data (GRxx)
3D0h–3D1h	Reserved	Reserved
2D Registers		
3D4h	VGA CRTC Index (CRX)	VGA CRTC Index (CRX)
3D5h	VGA CRTC Data (CRxx)	VGA CRTC Data (CRxx)
System Configuration Registers		

Address	Register Name (Read)	Register Name (Write)
3D6h	GFX/2D Configurations Extensions Index (XRX)	GFX/2D Configurations Extensions Index (XRX)
3D7h	GFX/2D Configurations Extensions Data (XRxx)	GFX/2D Configurations Extensions Data (XRxx)
2D Registers		
3D8h–3D9h	Reserved	Reserved
3DAh	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3DBh–3DFh	Reserved	Reserved

Indirect VGA and Extended VGA Register Indices

The registers listed in this section are indirectly accessed by programming an index value into the appropriate SRX, GRX, ARX, or CRX register. The index and data register address locations are listed in the previous section. Additional details concerning the indirect access mechanism are provided in the *VGA and Extended VGA Register Description* Chapter (see SRxx, GRxx, ARxx or CRxx sections).

2D Sequence Registers (3C4h / 3C5h)

Index	Sym	Description
00h	SR00	Sequencer Reset
01h	SR01	Clocking Mode
02h	SR02	Plane / Map Mask
03h	SR03	Character Font
04h	SR04	Memory Mode
07h	SR07	Horizontal Character Counter Reset

2D Graphics Controller Registers (3CEh / 3CFh)

Index	Sym	Register Name
00h	GR00	Set / Reset
01h	GR01	Enable Set / Reset
02h	GR02	Color Compare
03h	GR03	Data Rotate
04h	GR04	Read Plane Select
05h	GR05	Graphics Mode
06h	GR06	Miscellaneous
07h	GR07	Color Don’t Care
08h	GR08	Bit Mask
10h	GR10	Address Mapping
11h	GR11	Page Selector

Index	Sym	Register Name
18h	GR18	Software Flags

2D Attribute Controller Registers (3C0h / 3C1h)

Index	Sym	Register Name
00h	AR00	Palette Register 0
01h	AR01	Palette Register 1
02h	AR02	Palette Register 2
03h	AR03	Palette Register 3
04h	AR04	Palette Register 4
05h	AR05	Palette Register 5
06h	AR06	Palette Register 6
07h	AR07	Palette Register 7
08h	AR08	Palette Register 8
09h	AR09	Palette Register 9
0Ah	AR0A	Palette Register A
0Bh	AR0B	Palette Register B
0Ch	AR0C	Palette Register C
0Dh	AR0D	Palette Register D
0Eh	AR0E	Palette Register E
0Fh	AR0F	Palette Register F
10h	AR10	Mode Control
11h	AR11	Overscan Color
12h	AR12	Memory Plane Enable
13h	AR13	Horizontal Pixel Panning
14h	AR14	Color Select

2D CRT Controller Registers (3B4h / 3D4h / 3B5h / 3D5h)

Index	Sym	Register Name
00h	CR00	Horizontal Total
01h	CR01	Horizontal Display Enable End
02h	CR02	Horizontal Blanking Start
03h	CR03	Horizontal Blanking End
04h	CR04	Horizontal Sync Start
05h	CR05	Horizontal Sync End
06h	CR06	Vertical Total
07h	CR07	Overflow

Index	Sym	Register Name
08h	CR08	Preset Row Scan
09h	CR09	Maximum Scan Line
0Ah	CR0A	Text Cursor Start
0Bh	CR0B	Text Cursor End
0Ch	CR0C	Start Address High
0Dh	CR0D	Start Address Low
0Eh	CR0E	Text Cursor Location High
0Fh	CR0F	Text Cursor Location Low
10h	CR10	Vertical Sync Start
11h	CR11	Vertical Sync End
12h	CR12	Vertical Display Enable End
13h	CR13	Offset
14h	CR14	Underline Location
15h	CR15	Vertical Blanking Start
16h	CR16	Vertical Blanking End
17h	CR17	CRT Mode
18h	CR18	Line Compare
22h	CR22	Memory Read Latch Data

Memory Object Overview

Any memory data accessed by the device is considered part of a *memory object* of some memory object type.

The following table lists the various memory objects types and an indication of their role in the system.

Memory Object Type	Role
Graphics Translation Table (GTT)	Contains PTEs used to translate "graphics addresses" into physical memory addresses.
Hardware Status Page	Cached page of sysmem used to provide fast driver synchronization.
Logical Context Buffer	Memory areas used to store (save/restore) images of hardware rendering contexts. Logical contexts are referenced via a pointer to the corresponding Logical Context Buffer.
Ring Buffers	Buffers used to transfer (DMA) instruction data to the device. Primary means of controlling rendering operations.
Batch Buffers	Buffers of instructions invoked indirectly from Ring Buffers.
State Descriptors	Contains state information in a prescribed layout format to be read by hardware. Many different state descriptor formats are supported.
Vertex Buffers	Buffers of 3D vertex data indirectly referenced through "indexed" 3D primitive instructions.
VGA Buffer (Must be mapped UC on PCI)	Graphics memory buffer used to drive the display output while in legacy VGA mode.
Display Surface	Memory buffer used to display images on display devices.
Overlay Surface	Memory buffer used to display overlaid images on display devices.
Overlay Register, Filter Coefficients	Memory area used to provide double-buffer for Overlay register and filter coefficient loading.
Cursor Surface	Hardware cursor pattern in memory.
2D Render Source	Surface used as primary input to 2D rendering operations.
2D Render R-M-W Destination	2D rendering output surface that is read in order to be combined in the rendering function. Destination surfaces that accessed via this Read-Modify-Write mode have somewhat different restrictions than Write-Only Destination surfaces.
2D Render Write-Only Destination	2D rendering output surface that is written but not read by the 2D rendering function. Destination surfaces that accessed via a Write-Only mode have somewhat different restrictions than Read-Modify-Write Destination surfaces.
2D Monochrome Source	1 bpp surfaces used as inputs to 2D rendering after being converted to foreground/background colors.
2D Color Pattern	8x8 pixel array used to supply the "pattern" input to 2D rendering functions.
DIB	"Device Independent Bitmap" surface containing "logical" pixel values that are converted (via LUTs) to physical colors.

Memory Object Type	Role
3D Color Buffer	Surface receiving color output of 3D rendering operations. May also be accessed via R-M-W (aka blending). Also referred to as a Render Target.
3D Depth Buffer	Surface used to hold per-pixel depth and stencil values used in 3D rendering operations. Accessed via RMW.
3D Texture Map	Color surface (or collection of surfaces) which provide texture data in 3D rendering operations.
"Non-3D" Texture	Surface read by Texture Samplers, though not in normal 3D rendering operations (for example, in video color conversion functions).
Motion Comp Surfaces	These are the Motion Comp reference pictures.
Motion Comp Correction Data Buffer	This is Motion Comp intra-coded or inter-coded correction data.

Hardware Status Page

The hardware status page is a naturally-aligned 4KB page residing in snooped system memory. This page exists primarily to allow the device to report status via PCI master writes – thereby allowing the driver to read/poll WB memory instead of UC reads of device registers or UC memory.

The address of this page is programmed via the HWS_PGA MI register. The definition of that register (in *Memory Interface Registers*) includes a description of the layout of the Hardware Status Page.

Memory Access Indirection

The GPE supports the indirection of certain graphics (GTT-mapped) memory accesses. This support comes in the form of two *base address* state variables used in certain memory address computations with the GPE.

The intent of this functionality is to support the dynamic relocation of certain driver-generated memory structures after command buffers have been generated but prior to the their submittal for execution. For example, as the driver builds the command stream it could append pipeline state descriptors, kernel binaries, etc. to a general state buffer. References to the individual items would be inserted in the command buffers as offsets from the base address of the state buffer. The state buffer could then be freely relocated prior to command buffer execution, with the driver only needing to specify the final base address of the state buffer. Two base addresses are provided to permit surface-related state (binding tables, surface state tables) to be maintained in a state buffer separate from the general state buffer.

While the use of these base addresses is unconditional, the indirection can be effectively disabled by setting the base addresses to zero. The following table lists the various GPE memory access paths and which base address (if any) is relevant.

Base Address Utilization

Base Address Used	Memory Accesses
General State Base Address	**DataPort Read/Write DataPort memory accesses resulting from 'stateless' DataPort Read/Write requests. See** *DataPort* **for a definition of the 'stateless' form of requests.**
Dynamic State Base Address	Sampler reads of SAMPLER_STATE data and associated SAMPLER_BORDER_COLOR_STATE.
	Viewport states used by CLIP, SF, and WM/CC
	COLOR_CALC_STATE, DEPTH_STENCIL_STATE, and BLEND_STATE
	Push Constants (depending on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)
Instruction Base Address	**Normal EU instruction stream** (non-system routine)
	System routine EU instruction stream (starting address = SIP)
Surface State Base Address	Sampler and DataPort reads of BINDING_TABLE_STATE, as referenced by BT pointers passed via 3DSTATE_BINDING_TABLE_POINTERS
	Sampler and DataPort reads of SURFACE_STATE data
Indirect Object Base Address	**MEDIA_OBJECT Indirect Data** accessed by the CS unit .
None	CS unit reads from **Ring Buffers**, **Batch Buffers**
	CS writes resulting from PIPE_CONTROL command
	All VF unit memory accesses (**Index Buffers**, **Vertex Buffers**)
	All Sampler **Surface Memory Data** accesses (texture fetch, etc.)
	All **DataPort memory accesses**except 'stateless' DataPort Read/Write requests (e.g., RT accesses.) See *DataPort* for a definition of the 'stateless' form of requests.
	Memory reads resulting from **STATE_PREFETCH** commands
	Any **physical memory access** by the device
	GTT-mapped accesses not included above (i.e., default)
	Push Constants (depeding on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)

The following notation is used in the BSpec to distinguish between addresses and offsets:

Notation	Definition
PhysicalAddress[n:m]	Corresponding bits of a physical graphics memory byte address (not mapped by a GTT)
GraphicsAddress[n:m]	Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT)
GeneralStateOffset[n:m]	Corresponding bits of a relative byte offset added to the General State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
DynamicStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Dynamic State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)

Notation	Definition
InstructionBaseOffset[n:m]	Corresponding bits of a relative byte offset added to the Instruction Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
SurfaceStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Surface State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)

Instruction Ring Buffers

Instruction ring buffers are the memory areas used to pass instructions to the device. Refer to the Programming Interface chapter for a description of how these buffers are used to transport instructions.

The RINGBUF register sets (defined in Memory Interface Registers) are used to specify the ring buffer memory areas. The ring buffer must start on a 4KB boundary and be allocated in linear memory. The length of any one ring buffer is limited to 2MB.

Programming Note	
Context:	Instruction Ring Buffers in memory areas.
"Indirect" 3D primitive instructions (those that access vertex buffers) must reside in the same memory space as the vertex buffers.	

Instruction Batch Buffers

Instruction batch buffers are contiguous streams of instructions referenced via an MI_BATCH_BUFFER_START and related instructions (see Memory Interface Instructions, Programming Interface). They are used to transport instructions external to ring buffers.

Programming Note	
Context:	Instruction batch buffers in memory objects
Batch buffers can be tagged with any memory type when produced by IA. If WB memory type is used, it should be tagged with "snoop required" for GPU consumption (to trigger snoop from CPU cache).	

Programming Note	
Context:	Instruction batch buffers in memory objects
The batch buffer must be QWord aligned and a multiple of QWords in length. The ending address is the address of the last valid QWord in the buffer. The length of any single batch buffer is "virtually unlimited" (i.e., could theoretically be 4GB in length).	

Logical Contexts

This section is the lead section for the following subsections:

- BSD Logical Render Context Address (LRCA)
- Video Enhancement Logical Context Data
- Logical Contexts

BSD Logical Render Context Address (LRCA)

This section discusses the following topics for the BSD Logical Render Context Address (LRCA):

- Overall Context Layout
- Register State Context
- Ring Buffer
- Ring Context
- The Per-Process Hardware Status Page
- Video Engine Power Context

Overall Context Layout [

When Execlists are enabled, the Context Image for the media engine consists of two 4K pages:

- Per-Process HW Status Page (4K)
- Register State Context

When Execlists are disabled (ring buffer mode of scheduling) there is no concept of context for workloads submitted to the media engine, and hence there is no context image.

Register State context is explained in detail in the “Register State Context” Section.

Register/State Context

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)
ENGINE CONTEXT

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
NOOP		VCS	1	**0**
Load_Register_Immediate header	0x1100_101B	VCS	1	**0001**
Context Control	0x12244	VCS	2	**0002**
Ring Head Pointer Register	0x12034	VCS	2	**0004**
Ring Tail Pointer Register	0x12030	VCS	2	**0006**
RING_BUFFER_START	0x12038	VCS	2	**0008**
RING_BUFFER_CONTROL	0x1203C	VCS	2	**000A**
Batch Buffer Current Head Register (UDW)	0x12168	VCS	2	**000C**
Batch Buffer Current Head Register	0x12140	VCS	2	**000E**
Batch Buffer State Register	0x12110	VCS	2	**0010**
SECOND_BB_ADDR_UDW	0x1211C	VCS	2	**0012**
SECOND_BB_ADDR	0x12114	VCS	2	**0014**
SECOND_BB_STATE	0x12118	VCS	2	**0016**
BB_PER_CTX_PTR	0x121C0	VCS	2	**0018**
VCS_INDIRECT_CTX	0x121C4	VCS	2	**001A**
VCS_INDIRECT_CTX_OFFSET	0x121C8	VCS	2	**001C**
NOOP		VCS	2	**001E**
NOOP		VCS	1	**0020**
Load_Register_Immediate header	0x1100_1011	VCS	1	**0021**
CTX_TIMESTAMP	0x123A8	VCS	2	**0022**
PDP3_UDW	0x1228C	VCS	2	**0024**
PDP3_LDW	0x12288	VCS	2	**0026**
PDP2_UDW	0x12284	VCS	2	**0028**
PDP2_LDW	0x12280	VCS	2	**002A**
PDP1_UDW	0x1227C	VCS	2	**002C**
PDP1_LDW	0x12278	VCS	2	**002E**
PDP0_UDW	0x12274	VCS	2	**0030**
PDP0_LDW	0x12270	VCS	2	**0032**
NOOP		VCS	12	**0034**
NOOP		VCS	16	**0040**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
NOOP		VCS	1	**0050**
Load_Register_Immediate header	0x1100_1057	VCS	1	**0051**
EXCC	0x12028	VCS	2	**0052**
MI_MODE	0x1209C	VCS	2	**0054**
INSTPM	0x120C0	VCS	2	**0056**
PR_CTR_CTL	0x12178	VCS	2	**0058**
PR_CTR_THRSH	0x1217C	VCS	2	**005A**
TIMESTAMP Register (LSB)	0x12358	VCS	2	**005C**
BB_START_ADDR_UDW	0x12170	VCS	2	**005E**
BB_START_ADDR	0x12150	VCS	2	**0060**
BB_ADD_DIFF	0x12154	VCS	2	**0062**
BB_OFFSET	0x12158	VCS	2	**0064**
MI_PREDICATE_RESULT_1	0x1241C	VCS	2	**0066**
CS_GPR (1-16)	0x12600	VCS	64	**0068**
IPEHR	0x12068	VCS	2	**00A8**
NOOP		VCS	6	**00AA**

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in the Execution List mode of scheduling.

Programming Note	
Context:	Ring buffers - ring context area of LRCA
Ring Buffer registers are directly programmed in the Ring Buffer mode of scheduling.	

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution.
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

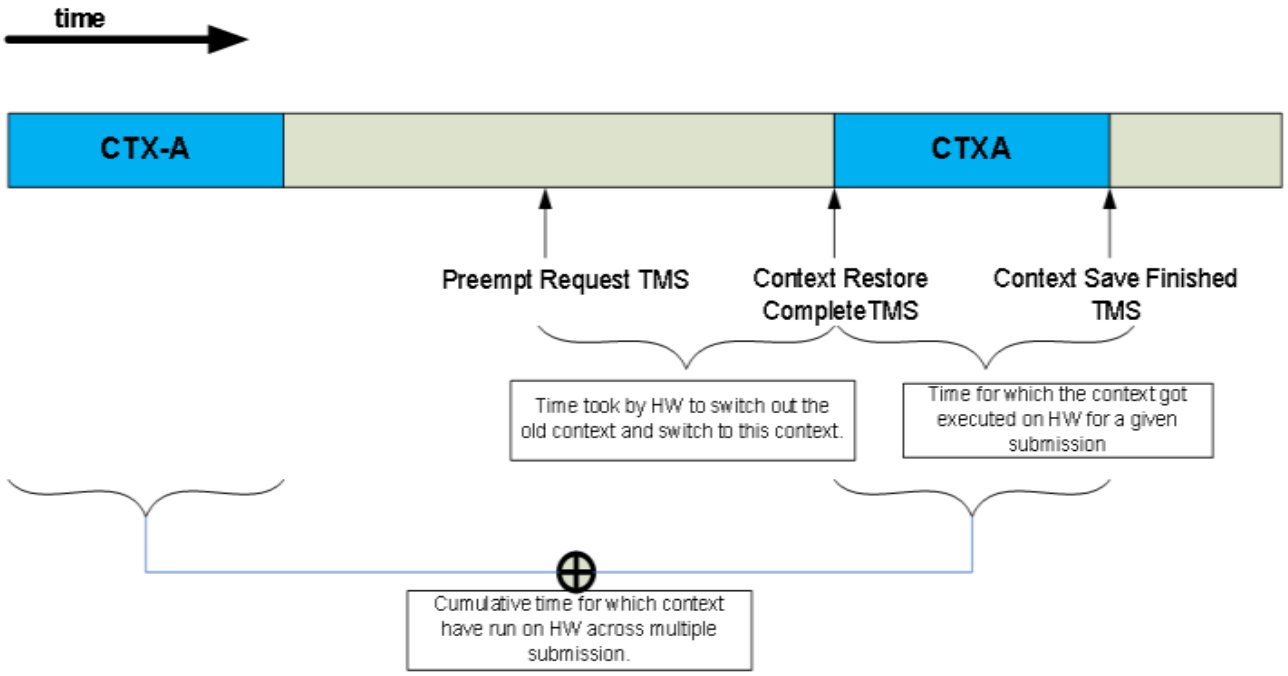
EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

The Per-Process Hardware Status

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

The figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Video Engine Power Context

The table below captures the data from VCS power context save/restored by PM. Address offset in the below table is relative to the starting location of VCS in the overall power context image managed by PM.

VCS Power Context Image

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(PWR)	CSFE/CSBE
NOOP		VCS	1	**0**	**CSFE**
Load_Register_Immediate header	0x1100_108B	VCS	1	**0001**	**CSFE**
GFX_MODE	0x1229C	VCS	2	**0002**	**CSFE**
GHWSP	0x12080	VCS	2	**0004**	**CSFE**
RING_BUFFER_CONTROL (Ring Always Disabled)	0x1203C	VCS	2	**0006**	**CSFE**
Ring Head Pointer Register	0x12034	VCS	2	**0008**	**CSFE**
Ring Tail Pointer Register	0x12030	VCS	2	**000A**	**CSFE**
RING_BUFFER_START	0x12038	VCS	2	**000C**	**CSFE**
RING_BUFFER_CONTROL (Original status)	0x1203C	VCS	2	**000E**	**CSFE**
Batch Buffer Current Head Register (UDW)	0x12168	VCS	2	**0010**	**CSFE**
Batch Buffer Current Head Register	0x12140	VCS	2	**0012**	**CSFE**
Batch Buffer State Register	0x12110	VCS	2	**0014**	**CSFE**
SECOND_BB_ADDR_UDW	0x1211C	VCS	2	**0016**	**CSFE**
SECOND_BB_ADDR	0x12114	VCS	2	**0018**	**CSFE**
SECOND_BB_STATE	0x12118	VCS	2	**001A**	**CSFE**
RC_PSMI_CONTROL	0x12050	VCS	2	**001C**	**CSFE**
RC_PWRCTX_MAXCNT	0x12054	VCS	2	**001E**	**CSFE**
CTX_WA_PTR	0x12058	VCS	2	**0020**	**CSFE**
NOPID	0x12094	VCS	2	**0022**	**CSFE**
HWSTAM	0x12098	VCS	2	**0024**	**CSFE**
IMR	0x120A8	VCS	2	**0026**	**CSFE**
EIR	0x120B0	VCS	2	**0028**	**CSFE**
EMR	0x120B4	VCS	2	**002A**	**CSFE**
CMD_CCTL_0	0x120C4	VCS	2	**002C**	**CSFE**
UHPTR	0x12134	VCS	2	**002E**	**CSFE**
BB_PREEMPT_ADDR_UDW	0x1216C	VCS	2	**0030**	**CSFE**
BB_PREEMPT_ADDR	0x12148	VCS	2	**0032**	**CSFE**
RING_BUFFER_HEAD_PREEMPT_REG	0x1214C	VCS	2	**0034**	**CSFE**
PREEMPT_DLY	0x12214	VCS	2	**0036**	**CSFE**
SYNC_FLIP_STATUS	0x122D0	VCS	2	**003A**	**CSFE**
SYNC_FLIP_STATUS_1	0x122D4	VCS	2	**003C**	**CSFE**
SYNC_FLIP_STATUS_2	0x122EC	VCS	2	**003E**	**CSFE**
WAIT_FOR_RC6_EXIT	0x120CC	VCS	2	**0040**	**CSFE**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(PWR)	CSFE/CSBE
RCS_CTXID_PREEMPTION_HINT	0x124CC	VCS	2	**0042**	**CSFE**
CS_PREEMPTION_HINT_UDW	0x124C8	VCS	2	**0044**	**CSFE**
CS_PREEMPTION_HINT	0x124BC	VCS	2	**0046**	**CSFE**
CCID Register	0x12180	VCS	2	**0048**	**CSFE**
SBB_PREEMPT_ADDRESS_UDW	0x12138	VCS	2	**004A**	**CSFE**
SBB_PREEMPT_ADDRESS	0x1213C	VCS	2	**004C**	**CSFE**
MI_PREDICATE_RESULT_2	0x123BC	VCS	2	**004E**	**CSFE**
CTXT_ST_PTR	0x123A0	VCS	2	**0050**	**CSFE**
CTXT_ST_BUF	0x12370	VCS	24	**0052**	**CSFE**
SEMA_WAIT_POLL	0x1224C	VCS	2	**006A**	**CSFE**
IDLEDELAY	0x1223C	VCS	2	**006C**	**CSFE**
DISPLAY MESSAGE FORWARD STATUS	0x122E8	VCS	2	**006E**	**CSFE**
RCS_FORCE_TO_NONPRIV	0x124D0	VCS	24	**0070**	**CSFE**
EXECLIST_STATUS_REGISTER	0x12234	VCS	2	**0088**	**CSFE**
CXT_OFFSET	0x21AC	VCS	2	**008C**	**CSBE**
NOOP			2	**008C**	**CSFE**
NOOP		VCS	1	**008E**	**CSBE**
Load_Register_Immediate header	0x1100_1005	VCS	1	**008F**	**CSBE**
GAC MODE REGISTER	0x120a0	VCS	2	**0094**	**CSBE**
NOOP		VCS	6	**0096**	**CSBE**
NOOP		VCS	1	**009C**	**CSBE**
MI_BATCH_BUFFER_END		VCS	1	**009D**	**CSBE**

Copy Engine Logical Context Data

Context Layout

When Execlists are enabled, the Context Image for the copy engine consists of two 4K pages:

Per-Process HW Status Page (4K)
Register State Context

When Execlists are disabled (ring buffer mode of scheduling) there is no concept of context for workloads submitted to media engine and hence there is no context image.

Register State context is explained in detail in “Register State Context” Section.

Register/State Context

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)
ENGINE CONTEXT

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
NOOP		BCS	1	**0**
Load_Register_Immediate header	0x1100_101B	BCS	1	**0001**
Context Control	0x22244	BCS	2	**0002**
Ring Head Pointer Register	0x22034	BCS	2	**0004**
Ring Tail Pointer Register	0x22030	BCS	2	**0006**
RING_BUFFER_START	0x22038	BCS	2	**0008**
RING_BUFFER_CONTROL	0x2203C	BCS	2	**000A**
Batch Buffer Current Head Register (UDW)	0x22168	BCS	2	**000C**
Batch Buffer Current Head Register	0x22140	BCS	2	**000E**
Batch Buffer State Register	0x22110	BCS	2	**0010**
SECOND_BB_ADDR_UDW	0x2211C	BCS	2	**0012**
SECOND_BB_ADDR	0x22114	BCS	2	**0014**
SECOND_BB_STATE	0x22118	BCS	2	**0016**
BB_PER_CTX_PTR	0x221C0	BCS	2	**0018**
BCS_INDIRECT_CTX	0x221C4	BCS	2	**001A**
BCS_INDIRECT_CTX_OFFSET	0x221C8	BCS	2	**001C**
NOOP		BCS	2	**001E**
NOOP		BCS	1	**0020**
Load_Register_Immediate header	0x1100_1011	BCS	1	**0021**
CTX_TIMESTAMP	0x223A8	BCS	2	**0022**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
PDP3_UDW	0x2228C	BCS	2	**0024**
PDP3_LDW	0x22288	BCS	2	**0026**
PDP2_UDW	0x22284	BCS	2	**0028**
PDP2_LDW	0x22280	BCS	2	**002A**
PDP1_UDW	0x2227C	BCS	2	**002C**
PDP1_LDW	0x22278	BCS	2	**002E**
PDP0_UDW	0x22274	BCS	2	**0030**
PDP0_LDW	0x22270	BCS	2	**0032**
NOOP		BCS	12	**0034**
NOOP		BCS	1	**0040**
Load_Register_Immediate header	0x1100_0001	BCS	1	**0041**
BCS TILE REGISTER	0x22200	BCS	2	**0042**
NOOP		BCS	12	**0044**
NOOP		BCS	1	**0050**
Load_Register_Immediate header	0x1100_1057	BCS	1	**0051**
EXCC	0x22028	BCS	2	**0052**
MI_MODE	0x2209C	BCS	2	**0054**
INSTPM	0x220C0	BCS	2	**0056**
PR_CTR_CTL	0x22178	BCS	2	**0058**
PR_CTR_THRSH	0x2217C	BCS	2	**005A**
TIMESTAMP Register (LSB)	0x22358	BCS	2	**005C**
BB_START_ADDR_UDW	0x22170	BCS	2	**005E**
BB_START_ADDR	0x22150	BCS	2	**0060**
BB_ADD_DIFF	0x22154	BCS	2	**0062**
BB_OFFSET	0x22158	BCS	2	**0064**
MI_PREDICATE_RESULT_1	0x2241C	BCS	2	**0066**
CS_GPR (1-16)	0x22600	BCS	64	**0068**
IPEHR	0x22068	BCS	2	**00A8**
NOOP		BCS	6	**00AA**
			DW	176
			K Bytes	0.6875

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in

Execution List mode of scheduling. Ring Buffer registers are directly programmed in Ring Buffer mode of scheduling.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

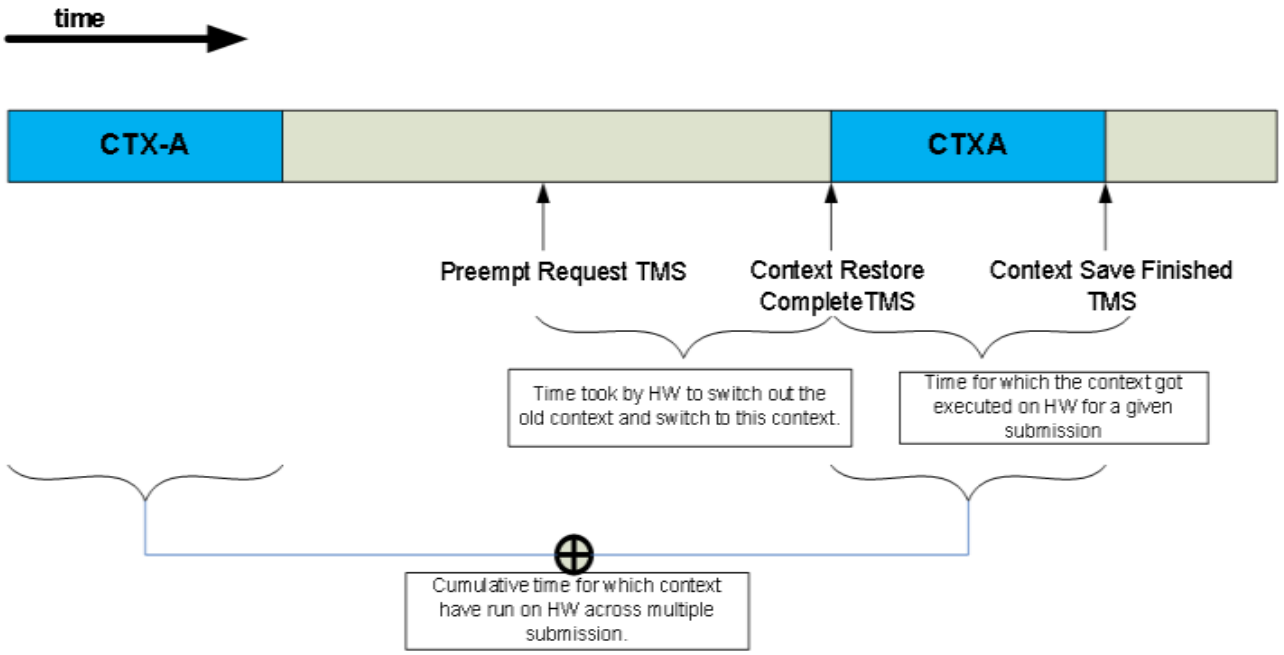
EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

The figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Blitter Engine Power Context

The table below captures the data from BCS power context save/restored by PM. Address offset in the below table is relative to the starting location of BCS in the overall power context image managed by PM.

BCS Power Context Image

Description			# of DW	Address Offset(PWR)	CSFE/CSBE
NOOP		BCS	1	**0**	**CSFE**
Load_Register_Immediate header	0x1100_108B	BCS	1	**0001**	**CSFE**
GFX_MODE	0x2229C	BCS	2	**0002**	**CSFE**
GHWSP	0x22080	BCS	2	**0004**	**CSFE**
RING_BUFFER_CONTROL (Ring Always Disabled)	0x2203C	BCS	2	**0006**	**CSFE**
Ring Head Pointer Register	0x22034	BCS	2	**0008**	**CSFE**
Ring Tail Pointer Register	0x22030	BCS	2	**000A**	**CSFE**
RING_BUFFER_START	0x22038	BCS	2	**000C**	**CSFE**
RING_BUFFER_CONTROL (Original status)	0x2203C	BCS	2	**000E**	**CSFE**
Batch Buffer Current Head Register (UDW)	0x22168	BCS	2	**0010**	**CSFE**
Batch Buffer Current Head Register	0x22140	BCS	2	**0012**	**CSFE**
Batch Buffer State Register	0x22110	BCS	2	**0014**	**CSFE**
SECOND_BB_ADDR_UDW	0x2211C	BCS	2	**0016**	**CSFE**
SECOND_BB_ADDR	0x22114	BCS	2	**0018**	**CSFE**
SECOND_BB_STATE	0x22118	BCS	2	**001A**	**CSFE**
RC_PSMI_CONTROL	0x22050	BCS	2	**001C**	**CSFE**
RC_PWRCTX_MAXCNT	0x22054	BCS	2	**001E**	**CSFE**
CTX_WA_PTR	0x22058	BCS	2	**0020**	**CSFE**
NOPID	0x22094	BCS	2	**0022**	**CSFE**
HWSTAM	0x22098	BCS	2	**0024**	**CSFE**
IMR	0x220A8	BCS	2	**0026**	**CSFE**
EIR	0x220B0	BCS	2	**0028**	**CSFE**
EMR	0x220B4	BCS	2	**002A**	**CSFE**
CMD_CCTL_0	0x220C4	BCS	2	**002C**	**CSFE**
UHPTR	0x22134	BCS	2	**002E**	**CSFE**
BB_PREEMPT_ADDR_UDW	0x2216C	BCS	2	**0030**	**CSFE**

Description			# of DW	Address Offset(PWR)	CSFE/CSBE
BB_PREEMPT_ADDR	0x22148	BCS	2	**0032**	**CSFE**
RING_BUFFER_HEAD_PREEMPT_REG	0x2214C	BCS	2	**0034**	**CSFE**
PREEMPT_DLY	0x22214	BCS	2	**0036**	**CSFE**
SYNC_FLIP_STATUS	0x222D0	BCS	2	**003A**	**CSFE**
SYNC_FLIP_STATUS_1	0x222D4	BCS	2	**003C**	**CSFE**
SYNC_FLIP_STATUS_2	0x222EC	BCS	2	**003E**	**CSFE**
WAIT_FOR_RC6_EXIT	0x220CC	BCS	2	**0040**	**CSFE**
RCS_CTXID_PREEMPTION_HINT	0x224CC	BCS	2	**0042**	**CSFE**
CS_PREEMPTION_HINT_UDW	0x224C8	BCS	2	**0044**	**CSFE**
CS_PREEMPTION_HINT	0x224BC	BCS	2	**0046**	**CSFE**
CCID Register	0x22180	BCS	2	**0048**	**CSFE**
SBB_PREEMPT_ADDRESS_UDW	0x22138	BCS	2	**004A**	**CSFE**
SBB_PREEMPT_ADDRESS	0x2213C	BCS	2	**004C**	**CSFE**
MI_PREDICATE_RESULT_2	0x223BC	BCS	2	**004E**	**CSFE**
CTXT_ST_PTR	0x223A0	BCS	2	**0050**	**CSFE**
CTXT_ST_BUF	0x22370	BCS	24	**0052**	**CSFE**
SEMA_WAIT_POLL	0x2224C	BCS	2	**006A**	**CSFE**
IDLEDELAY	0x2223C	BCS	2	**006C**	**CSFE**
DISPLAY MESSAGE FORWARD STATUS	0x222E8	BCS	2	**006E**	**CSFE**
RCS_FORCE_TO_NONPRIV	0x224D0	BCS	24	**0070**	**CSFE**
EXECLIST_STATUS_REGISTER	0x22234	BCS	2	**0088**	**CSFE**
CXT_OFFSET	0x21AC	BCS	2	**008C**	**CSBE**
NOOP		BCS	2	**008C**	**CSFE**
NOOP		BCS	1	**008E**	**CSBE**
Load_Register_Immediate header	0x1100_1003	BCS	1	**008F**	**CSBE**
GAB MODE REGISTER	0x220a0	BCS	2	**0092**	**CSBE**
NOOP		BCS	8	**0094**	**CSBE**
NOOP		BCS	1	**009C**	**CSBE**
MI_BATCH_BUFFER_END		BCS	1	**009D**	**CSBE**

Video Enhancement Logical Context Data

Context Layout

When Execlists are enabled, the Context Image for the Video Enhancement engine consists of two 4K pages:

- Per-Process HW Status Page (4K)
- Register State Context

When Execlists are disabled (ring buffer mode of scheduling) there is no concept of context for workloads submitted to media engine and hence there is no context image.

Register State context is explained in detail in the “Register State Context” Section.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

Register/State Context

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)
ENGINE CONTEXT

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
NOOP		VECS	1	**0**
Load_Register_Immediate header	0x1100_101B	VECS	1	**0001**
Context Control	0x1A244	VECS	2	**0002**
Ring Head Pointer Register	0x1A034	VECS	2	**0004**
Ring Tail Pointer Register	0x1A030	VECS	2	**0006**
RING_BUFFER_START	0x1A038	VECS	2	**0008**
RING_BUFFER_CONTROL	0x1A03C	VECS	2	**000A**
Batch Buffer Current Head Register (UDW)	0x1A168	VECS	2	**000C**
Batch Buffer Current Head Register	0x1A140	VECS	2	**000E**
Batch Buffer State Register	0x1A110	VECS	2	**0010**
SECOND_BB_ADDR_UDW	0x1A11C	VECS	2	**0012**
SECOND_BB_ADDR	0x1A114	VECS	2	**0014**
SECOND_BB_STATE	0x1A118	VECS	2	**0016**
BB_PER_CTX_PTR	0x1A1C0	VECS	2	**0018**
VECS_INDIRECT_CTX	0x1A1C4	VECS	2	**001A**
VECS_INDIRECT_CTX_OFFSET	0x1A1C8	VECS	2	**001C**
NOOP		VECS	2	**001E**
NOOP		VECS	1	**0020**
Load_Register_Immediate header	0x1100_1011	VECS	1	**0021**
CTX_TIMESTAMP	0x1A3A8	VECS	2	**0022**
PDP3_UDW	0x1A28C	VECS	2	**0024**
PDP3_LDW	0x1A288	VECS	2	**0026**
PDP2_UDW	0x1A284	VECS	2	**0028**
PDP2_LDW	0x1A280	VECS	2	**002A**
PDP1_UDW	0x1A27C	VECS	2	**002C**
PDP1_LDW	0x1A278	VECS	2	**002E**
PDP0_UDW	0x1A274	VECS	2	**0030**
PDP0_LDW	0x1A270	VECS	2	**0032**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset(Dword)
NOOP		VECS	12	**0034**
NOOP		VECS	16	**0040**
NOOP		VECS	1	**0050**
Load_Register_Immediate header	0x1100_1057	VECS	1	**0051**
EXCC	0x1A028	VECS	2	**0052**
MI_MODE	0x1A09C	VECS	2	**0054**
INSTPM	0x1A0C0	VECS	2	**0056**
PR_CTR_CTL	0x1A178	VECS	2	**0058**
PR_CTR_THRSH	0x1A17C	VECS	2	**005A**
TIMESTAMP Register (LSB)	0x1A358	VECS	2	**005C**
BB_START_ADDR_UDW	0x1A170	VECS	2	**005E**
BB_START_ADDR	0x1A150	VECS	2	**0060**
BB_ADD_DIFF	0x1A154	VECS	2	**0062**
BB_OFFSET	0x1A158	VECS	2	**0064**
MI_PREDICATE_RESULT_1	0x1A41C	VECS	2	**0066**
CS_GPR (1-16)	0x1A600	VECS	64	**0068**
IPEHR	0x1A068	VECS	2	**00A8**
NOOP		VECS	6	**00AA**

Video Enhancement Engine Power Context

The table below captures the data from VECS power context save/restored by PM. Address offset in the below table is relative to the starting location of VECS in the overall power context image managed by PM.

VECS Power Context Image

Description			# of DW	Address Offset(PWR)	CSFE/CSBE
NOOP		VECS	1	**0**	**CSFE**
Load_Register_Immediate header	0x1100_108B	VECS	1	**0001**	**CSFE**
GFX_MODE	0x1A29C	VECS	2	**0002**	**CSFE**
GHWSP	0x1A080	VECS	2	**0004**	**CSFE**
RING_BUFFER_CONTROL (Ring Always Disabled)	0x1A03C	VECS	2	**0006**	**CSFE**
Ring Head Pointer Register	0x1A034	VECS	2	**0008**	**CSFE**
Ring Tail Pointer Register	0x1A030	VECS	2	**000A**	**CSFE**
RING_BUFFER_START	0x1A038	VECS	2	**000C**	**CSFE**
RING_BUFFER_CONTROL (Original status)	0x1A03C	VECS	2	**000E**	**CSFE**
Batch Buffer Current Head Register (UDW)	0x1A168	VECS	2	**0010**	**CSFE**
Batch Buffer Current Head Register	0x1A140	VECS	2	**0012**	**CSFE**
Batch Buffer State Register	0x1A110	VECS	2	**0014**	**CSFE**
SECOND_BB_ADDR_UDW	0x1A11C	VECS	2	**0016**	**CSFE**
SECOND_BB_ADDR	0x1A114	VECS	2	**0018**	**CSFE**
SECOND_BB_STATE	0x1A118	VECS	2	**001A**	**CSFE**
RC_PSMI_CONTROL	0x1A050	VECS	2	**001C**	**CSFE**
RC_PWRCTX_MAXCNT	0x1A054	VECS	2	**001E**	**CSFE**
CTX_WA_PTR	0x1A058	VECS	2	**0020**	**CSFE**
NOPID	0x1A094	VECS	2	**0022**	**CSFE**
HWSTAM	0x1A098	VECS	2	**0024**	**CSFE**
IMR	0x1A0A8	VECS	2	**0026**	**CSFE**
EIR	0x1A0B0	VECS	2	**0028**	**CSFE**
EMR	0x1A0B4	VECS	2	**002A**	**CSFE**
CMD_CCTL_0	0x1A0C4	VECS	2	**002C**	**CSFE**
UHPTR	0x1A134	VECS	2	**002E**	**CSFE**
BB_PREEMPT_ADDR_UDW	0x1A16C	VECS	2	**0030**	**CSFE**
BB_PREEMPT_ADDR	0x1A148	VECS	2	**0032**	**CSFE**

Description			# of DW	Address Offset(PWR)	CSFE/CSBE
RING_BUFFER_HEAD_PREEMPT_REG	0x1A14C	VECS	2	**0034**	**CSFE**
PREEMPT_DLY	0x1A214	VECS	2	**0036**	**CSFE**
SYNC_FLIP_STATUS	0x1A2D0	VECS	2	**003A**	**CSFE**
SYNC_FLIP_STATUS_1	0x1A2D4	VECS	2	**003C**	**CSFE**
SYNC_FLIP_STATUS_2	0x1A2EC	VECS	2	**003E**	**CSFE**
WAIT_FOR_RC6_EXIT	0x1A0CC	VECS	2	**0040**	**CSFE**
RCS_CTXID_PREEMPTION_HINT	0x1A4CC	VECS	2	**0042**	**CSFE**
CS_PREEMPTION_HINT_UDW	0x1A4C8	VECS	2	**0044**	**CSFE**
CS_PREEMPTION_HINT	0x1A4BC	VECS	2	**0046**	**CSFE**
CCID Register	0x1A180	VECS	2	**0048**	**CSFE**
SBB_PREEMPT_ADDRESS_UDW	0x1A138	VECS	2	**004A**	**CSFE**
SBB_PREEMPT_ADDRESS	0x1A13C	VECS	2	**004C**	**CSFE**
MI_PREDICATE_RESULT_2	0x1A3BC	VECS	2	**004E**	**CSFE**
CTXT_ST_PTR	0x1A3A0	VECS	2	**0050**	**CSFE**
CTXT_ST_BUF	0x1A370	VECS	24	**0052**	**CSFE**
SEMA_WAIT_POLL	0x1A24C	VECS	2	**006A**	**CSFE**
IDLEDELAY	0x1A23C	VECS	2	**006C**	**CSFE**
DISPLAY MESSAGE FORWARD STATUS	0x1A2E8	CS	2	**006E**	**CSFE**
RCS_FORCE_TO_NONPRIV	0x1A4D0	CS	24	**0070**	**CSFE**
EXECLIST_STATUS_REGISTER	0x1A234	CS	2	**0088**	**CSFE**
CXT_OFFSET	0x21AC	CS	2	**008C**	**CSBE**
NOOP		VECS	2	**008C**	**CSFE**
NOOP		VECS	1	**008E**	**CSBE**
Load_Register_Immediate header	0x1100_1003	VECS	1	**008F**	**CSBE**
NOOP		VECS	8	**0094**	**CSBE**
NOOP		VECS	1	**009C**	**CSBE**
MI_BATCH_BUFFER_END		VECS	1	**009D**	**CSBE**

Memory Data Formats

This chapter describes the attributes associated with the memory-resident data objects operated on by the graphics pipeline. This includes object types, pixel formats, memory layouts, and rules/restrictions placed on the dimensions, physical memory location, pitch, alignment, etc. with respect to the specific operations performed on the objects.

Unsigned Normalized (UNORM)

An unsigned normalized value with *n* bits is interpreted as a value between 0.0 and 1.0. The minimum value (all 0's) is interpreted as 0.0, the maximum value (all 1's) is interpreted as 1.0. Values in between are equally spaced. For example, a 2-bit UNORM value would have the four values 0, 1/3, 2/3, and 1.

If the incoming value is interpreted as an n-bit integer, the interpreted value can be calculated by dividing the integer by 2^n-1.

Gamma Conversion (SRGB)

Gamma conversion is only supported on UNORM formats. If this flag is included in the surface format name, it indicates that a reverse gamma conversion is to be done after the source surface is read, and a forward gamma conversion is to be done before the destination surface is written.

Signed Normalized (SNORM)

Programming Note	
Context:	Signed normalized value in memory data formats.
A signed normalized value with *n* bits is interpreted as a value between -1 and +1.0. If the incoming value is interpreted as a 2's-complement n-bit integer, the interpreted value can be calculated by dividing the integer by 2^{n-1}-1. The most negative value of -2^{n-1} will result in a value slightly smaller than -1.0. This value is clamped to -1.0; thus, there are two representations of -1.0 in SNORM format.	

Unsigned Integer (UINT/USCALED)

The UINT and USCALED formats interpret the source as an unsigned integer value with *n* bits with a range of 0 to 2^n-1.

The UINT formats copy the source value to the destination (zero-extending if required), keeping the value as an integer.

The USCALED formats convert the integer into the corresponding floating point value (e.g., 0x03 --> 3.0f). For 32-bit sources, the value is rounded to nearest even.

Signed Integer (SINT/SSCALED)

A signed integer value with *n* bits is interpreted as a 2's complement integer with a range of -2^{n-1} to $+2^{n-1}-1$.

The SINT formats copy the source value to the destination (sign-extending if required), keeping the value as an integer.

The SSCALED formats convert the integer into the corresponding floating point value (e.g., 0xFFFD --> -3.0f). For 32-bit sources, the value is rounded to nearest even.

Floating Point (FLOAT)

Refer to IEEE Standard 754 for Binary Floating-Point Arithmetic. The IA-32 Intel (R) Architecture Software Developer's Manual also describes floating point data types .

64-bit Floating Point

Bit	Description
63	**Sign (s)**
62:52	**Exponent (e)** Biased Exponent
51:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == b'11..11' and f != 0, then v is NaN regardless of s
- if e == b'11..11' and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < b'11..11', then v = $(-1)^s*2^{(e-1023)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-1022)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s$*0 (signed zero)

32-bit Floating Point

Bit	Description
31	**Sign (s)**
30:23	**Exponent (e)** Biased Exponent
22:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == 255 and f != 0, then v is NaN regardless of s
- if e == 255 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 255, then v = $(-1)^s*2^{(e-127)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-126)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s$*0 (signed zero)

16-bit Floating Point

Bit	Description
15	**Sign (s)**
14:10	**Exponent (e)** Biased Exponent
9:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == 31 and f != 0, then v is NaN regardless of s
- if e == 31 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 31, then v = $(-1)^s*2^{(e-15)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-14)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

The following table represents relationship between 32 bit and 16 bit floating point ranges:

flt32 exponent	Unbiased exponent	Normalization	flt16 exponent	flt16 fraction
255				
254	127			
...				
127+16	16	Infinity	31	1.1111111111
127+15	15	Max exponent	30	1.xxxxxxxxxx
127	0		15	1.xxxxxxxxxx
113	-14	Min exponent	1	1.xxxxxxxxxx
112		Denormalized	0	0.1xxxxxxxxx
111		Denormalized	0	0.01xxxxxxxx
110		Denormalized	0	0.001xxxxxxx
109		Denormalized	0	0.0001xxxxxx
108		Denormalized	0	0.00001xxxxx
107		Denormalized	0	0.000001xxxx
106		Denormalized	0	0.0000001xxx
115		Denormalized	0	0.00000001xx
114		Denormalized	0	0.000000001x
113		Denormalized	0	0.0000000001
112		Denormalized	0	0.0
...				
0			0	0.0

Conversion from the 32-bit floating point format to the 16-bit format should be done with round to nearest even.

11-bit Floating Point

Bits	Description
10:6	**Exponent (e):** Biased exponent (the bias depends on e)
5:0	**Fraction (f):** Fraction bits to the right of the binary point

The value v of an 11-bit floating-point number is calculated from e and f as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

There is no sign bit and negative values are not represented.

The 11-bit floating-point format has one more bit of fractional precision than the 10-bit floating-point format.

The maximum representable finite value is 1.111111b * 2^{15} = FE00h = 65024.

10-bit Floating Point

Bits	Description
9:5	**Exponent (e):** Biased exponent (the bias depends on e)
4:0	**Fraction (f):** Fraction bits to the right of the binary point

The value v of a 10-bit floating-point number is calculated from e and f as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

There is no sign bit and negative values are not represented.

The maximum representable finite value is 1.11111b * 2^{15} = FC00h = 64512.

Shared Exponent

The R9G9B9E5_SHAREDEXP format contains three channels that share an exponent. The three fractions assume an impled "0" rather than an implied "1" as in the other floating point formats. This format does not support infinity and NaN values. There are no sign bits, only positive numbers and zero can be represented. The value of each channel is determined as follows, where "f" is the fraction of the corresponding channel, and "e" is the shared exponent.

$v = (0.f)*2^{(e-15)}$

Bit	Description
31:27	**Exponent (e)** Biased Exponent
26:18	**Blue Fraction**
17:9	**Green Fraction**
8:0	**Red Fraction**

Memory Compression

Media Memory Compression

The software requirement when using media memory compression is to allocate each compressible surface one memory tile wider than is required based on the surface width plus normal byte padding (this approach is called "pitch+1"). The reason is each compressible surface needs an "extra" tile to the right edge of surface to store important compression control information. For example, if the surface is 1920x1088, this would normally be allocated by the driver to be 2048 bytes wide, or 16 tiles (for NV12 8bpp). Using this "pitch + 1", the pitch would be set to 17 instead of 16 (and the surface width remains unchanged, only pitch is increased).

The largest supported width will be 4K pixels for 2D RGBA 8bpp surfaces and 2x2K for S3D surfaces (for 4KB pages). E.g. the pitch would be set to 129 in these cases (128+1). NV12 4K would be 33 (28+1). The case of 64KB pages is the same: the driver will allocate 1 extra page to the right ("pitch + 1"), however now the 4K wide restriction is relaxed. With 64KB pages, the widest surface that supports memory compression is 16K for 2D RGBA 8bpp or 2x8K for S3D. E.g. the pitch would be set to 129 in these cases (128+1).

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 4: Configurations

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Configurations Overview 1

Top Level Block Diagrams 2

Device Attributes 6

Steppings and Device IDs 10

Configurations Overview

The Intel "Gen" Graphics Architecture was first introduced to the market in 2004. Since that time, the architecture and implementation have evolved to add many new features, increase performance, and improve power efficiency.

Each product generation has its own configurations chapter. Each chapter has a section for each project, and each project contains the following subsections:

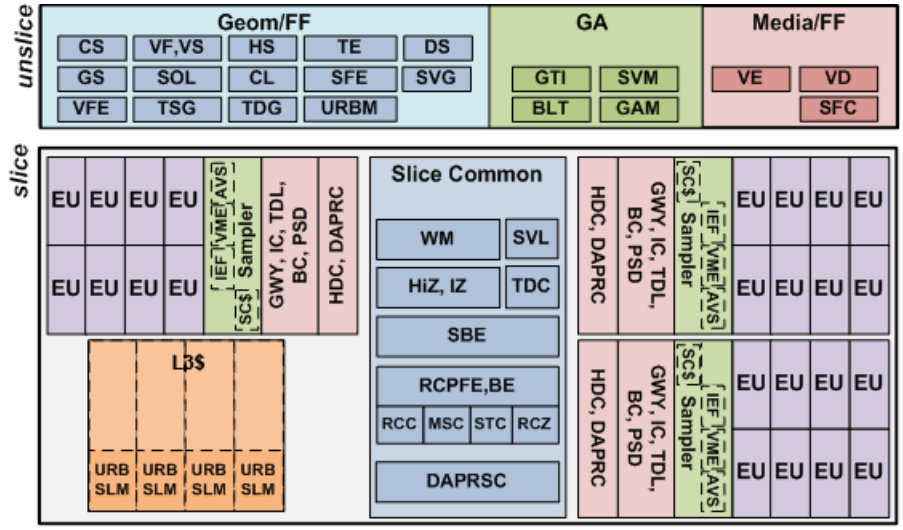
- Top Level Block Diagrams - Show basic feature blocks of the project's graphics architecture, for GT configurations.
- Device Attributes - List details of the graphics configuration options for each project.
- Steppings and Device IDs - Lists all of the current unique GT Die / Packages for a specific project.

Top Level Block Diagrams

The diagrams below show basic feature blocks of the Kaby Lake (KBL) graphics architecture.

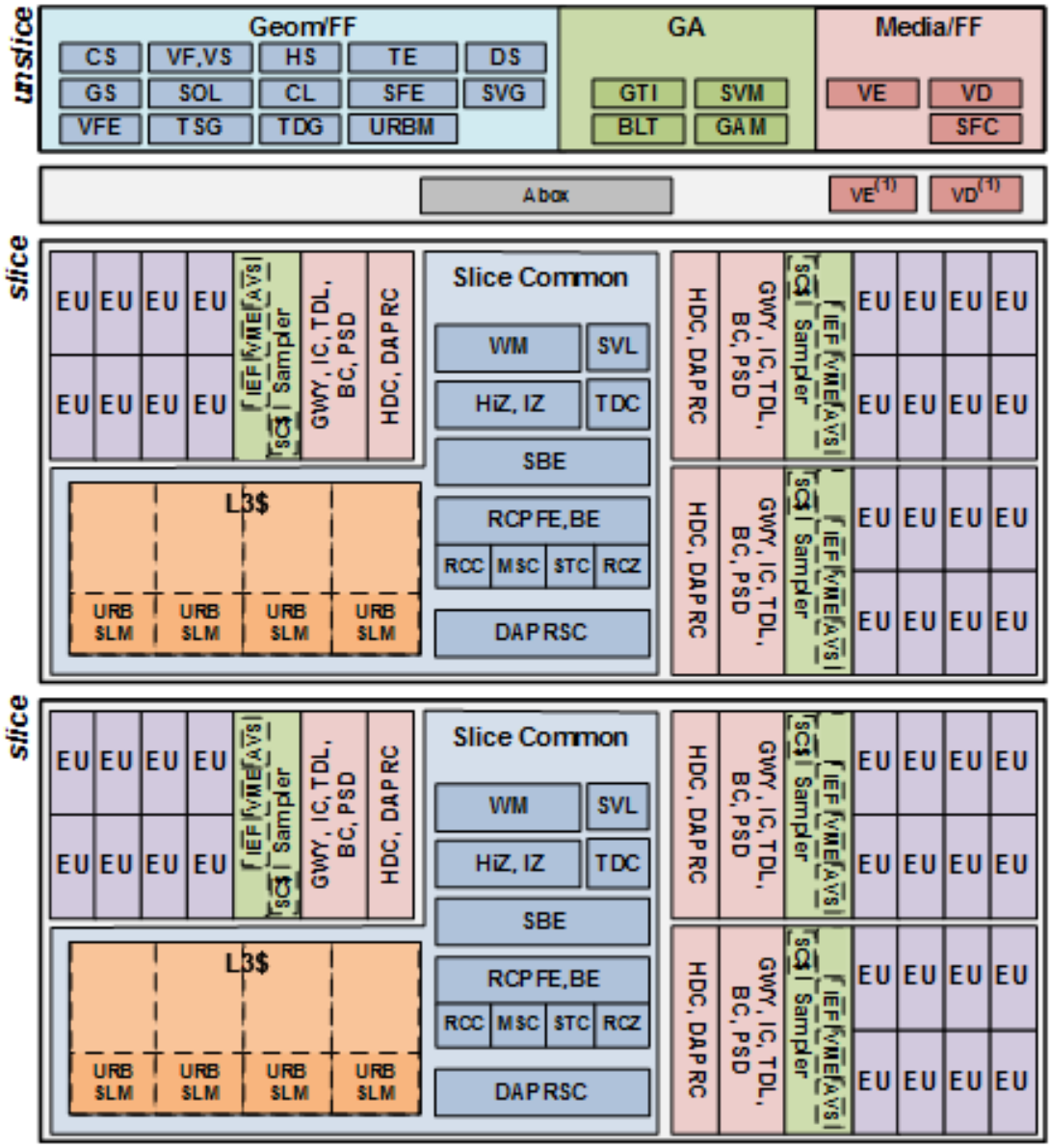
GT2 Configuration

The GT2 configuration contains one Unslice and one Slice with separate power domains for each, although they share a single clock domain.



GT3 Configuration

The GT3 configuration has an identical Unslice to GT2, except that it contains two Slices. Separate clock domains for the Unslice and Slice may be available depending on SKU. The L3 caches of each Slice combine to provide an aggregate L3 cache of twice the size and twice the bandwidth of a single instance. GT3 also has additional media blocks with second instance of VEBox and VDBox each.



This diagram is based on the following functional partitions:

(a) Geometry Fixed Functions (Geom/FF)

(b) Media Fixed Functions (Media/FF)

(c) Global Assets and GT Interface (GA)

(d) One or more Subslices (three shown)

(e) A Slice Common block

(f) An L3 Cache (L3$) block

Note that the combination of (a), (b), and (c) is typically referred to as the “unslice”, while a combination of (d), (e), and (f) is referred to as a compute “slice”.

The functionality in each of these groupings is further broken down as follows:

- Unslice – Fixed function pipelines for 3D, GPGPU, and Media operations, and interface to the outside world.
    - The 3D Geometry / Fixed Function (Geom/FF) block consisting of:
        - 3D fixed function pipeline (CS, VFVS, HS, TE, DS, GS, SOL, SL, SFE, SVG)
        - Video Front-End unit (VFE)
        - Thread Spawner unit (TSG) and the global Thread Dispatcher unit (TDG)
        - Unified Return Buffer Manager (URBM)
        -
    - Media fixed function assets:
        - Video Decode (VD) Box
        - Video Encode (VE) Box
        - Scaler & Format Converter (SFC)
    - The Global Assets (GA) block as the primary interface and memory stream gateway to the outside world, consisting of:
        - GT Interface (GTI)
        - State Variable Manager (SVM)
        - Blitter (BLT)
        - Graphics Arbiter (GAM)
- Subslice (three shown) – A compute unit with supporting fixed- or shared-function assets sufficient for the EU capability.
    - A bank of Execution Units (EUs) – eight per subslice shown
    - Sampler, supporting both media and 3D functions
    - Gateway (GWY)
    - Instruction cache (IC)
    - Local Thread Dispatcher (TDL)
    - Barycentric Calculator (BC)
    - Pixel Shader Dispatcher (PSD)
    - Data Cluster (HDC)
    - Dataport Render Cache (DAPRC) - two per subslice
- Slice Common – Scalable fixed function assets which support the compute horsepower provided two or more subslices.
    - 3D Fixed Function:
        - Windower/Mask unit (WM)
        - Hi-Z (HZ) and Intermediate Z (IZ)
        - Setup Backend (SBE)

        - RCPFE, BE
        - 3D stream caches (RCC, MSC, STC, RCZ)
    - Media Fixed Functions:
        - DAPRSC
        - SVL
        - TDC
- L3 Cache – backing L3 cache for certain memory streams emanating from subslices.
    - L3 Data cache with support for data, URB, and shared local memory (SLM)

Device Attributes

The following table lists detailed GT device attributes for proposed KBL SKUs.

NOTE: This information is preliminary, and subject to change.

Product Configuration Attribute Table				
Product Family	**KBL**			
Architectural Name *	**1x2x6**	**1x3x6**	**1x3x8**	**2x3x8**
SKU Name	**GT1F**	**GT1.5F**	**GT2**	**GT3**
Global Attributes				
Slice count	1	1	1	2
Subslice Count	2	3	3	6
EU/Subslice	6	6	8	8
EU count (total)	12	18	23 / 24 [b]	47 / 48 [b]
Thread Count	7	7	7	7
Thread Count (Total)	84	126	161 / 168	329 / 336
FLOPs/Clk - Half Precision, MAD (peak)	384	576	736 / 768	1504 / 1536
FLOPs/Clk - Single Precision, MAD (peak)	192	288	368 / 384	752 / 768
FLOPs/Clk - Double Precision, MAD (peak)	48	72	92 / 96	188 / 192
Unslice clocking (coupled/decoupled from Cr slice)	coupled	coupled	coupled	coupled
GTI / Ring Interfaces	1	1	1	1
GTI bandwidth (bytes/unslice-clk)	64: R	64: R	64: R	64: R
	64: W	64: W	64: W	64: W
eDRAM Support	N/A	N/A	N/A	0, 64MB
Graphics Virtual Address Range	48 bit	48 bit	48 bit	48 bit
Graphics Physical Address Range	39 bit	39 bit	39 bit	39 bit
Caches & Dedicated Memories				
L3 Cache, total size (bytes)	384K	768K	768K	1536K
L3 Cache, bank count	2	4	4	8
L3 Cache, bandwidth (bytes/clk)	2x 64: R 2x 64: W	4x 64: R 4x 64: W	4x 64: R 4x 64: W	8x 64: R 8x 64: W
L3 Cache, D$ Size (Kbytes)	192K - 256K	512K	512K	1024K
URB Size (kbytes)	128K - 192K	384K	384K	768K
SLM Size (kbytes)	0, 128K	0, 192K	0, 192K	0, 384K
LLC/L4 size (bytes) [1]	~2MB/CPU core	~2MB/CPU core	~2MB/CPU core	~2MB/CPU core
Instruction Cache (IC, bytes)	2x 48K	3x 48K	3x 48K	6x 48K
Color Cache (RCC, bytes)	24K	24K	24K	2x 24K

Product Configuration Attribute Table				
MSC Cache (MSC, bytes)	16K	16K	16K	2x 16K
HiZ Cache (HZC, bytes)	12K	12K	12K	2x 12K
Z Cache (RCZ, bytes)	32K	32K	32K	2x 32K
Stencil Cache (STC, bytes)	8K	8K	8K	2x 8K
Instruction Issue Rates				
FMAD, SP (ops/EU/clk)	8	8	8	8
FMUL, SP (ops/EU/clk)	8	8	8	8
FADD, SP (ops/EU/clk)	8	8	8	8
MIN,MAX, SP (ops/EU/clk)	8	8	8	8
CMP, SP (ops/EU/clk)	8	8	8	8
INV, SP (ops/EU/clk)	2	2	2	2
SQRT, SP (ops/EU/clk)	2	2	2	2
RSQRT, SP (ops/EU/clk)	2	2	2	2
LOG, SP (ops/EU/clk)	2	2	2	2
EXP, SP (ops/EU/clk)	2	2	2	2
POW, SP (ops/EU/clk)	1	1	1	1
IDIV, SP (ops/EU/clk)	1-6	1-6	1-6	1-6
TRIG, SP (ops/EU/clk)	2	2	2	2
FDIV, SP (ops/EU/clk)	1	1	1	1
Load/Store				
Data Ports (HDC)	2	3	3	6
L3 Load/Store (dwords/clk)	2x 64	3x 64	3x 64	6x 64
SLM Load/Store (dwords/clk)	2x 64	3x 64	3x 64	6x 64
Atomic Inc, 32b - sequential addresses (dwords/clk)	2x 64	3x 64	3x 64	6x 64
Atomic Inc, 32b - same address (dwords/clk)	2x 4	3x 4	3x 4	6x 4
Atomic CmpWr, 32b - sequential addresses (dwords/clk)	2x 32	3x 32	3x 32	6x 32
Atomic CmpWr, 32b - same address (dwords/clk)	2x 4	3x 4	3x 4	6x 4
3D Attributes				
Geometry pipes	1	1	1	1
Samplers (3D)	2	3	3	6
Texel Rate, point, 32b (tex/clk)	8	12	12	24
Texel Rate, point, 64b (tex/clk)	8	12	12	24
Texel Rate, point, 128b (tex/clk)	8	12	12	24
Texel Rate, bilinear, 32b (tex/clk)	8	12	12	24
Texel Rate, bilinear, 64b (tex/clk)	8	12	12	24

Product Configuration Attribute Table				
Texel Rate, bilinear, 128b (tex/clk)	2	3	3	6
Texel Rate, trilinear, 32b (tex/clk)	8	12	12	24
Texel Rate, trilinear, 64b (tex/clk)	4	6	6	12
Texel Rate, trilinear, 128b (tex/clk)	1	1.5	1.5	3
Texel Rate, aniso 2x, MIP Linear,, 32b (tex/clk)	2	3	3	6
Texel Rate, aniso 4x, MIP Linear,, 32b (tex/clk)	1	1.5	1.5	3
Texel Rate, ansio 8x, MIP Linear,, 32b (tex/clk)	0.5	0.75	0.75	1.5
Texel Rate, ansio 16x, MIP Linear,, 32b (tex/clk)	0.25	0.375	0.375	0.75
HiZ Rate, (ppc)	64	64	64	2x 64
IZ Rate, (ppc)	16	16	16	2x 16
Stencil Rate (ppc)	64	64	64	2x 64
(500 MHz, DDR-2400 or eDRAM; Range depends on dynamic compression ratio)				
Pixel Rate, fill, 32bpp (pix/clk, RCC hit)	8	8	8	16
Pixel Rate, fill, 32bpp (pix/clk, LLC hit @ 1.0x unslice clk) [2]				
Pixel Rate, fill, 32bpp (pix/clk, LLC hit, @ 1.5x unslice clk) [2]	N/A	N/A	N/A	
Pixel Rate, fill, 32bpp (pix/clk, memory, @ 1.0x unslice clk) [2]				
Pixel Rate, fill, 32bpp (pix/clk, memory, @ 1.5x unslice clk) [2]	N/A	N/A	N/A	
(500 MHz, DDR-2400 or eDRAM; Range depends on dynamic compression ratio)				
Pixel Rate, blend, 32bpp (p/clk, RCC hit)	8	8	8	16
Pixel Rate, blend, 32bpp (p/clk, LLC hit, @ 1.0x unslice clk) [2]				
Pixel Rate, blend, 32bpp (p/clk, LLC hit, @ 1.5x unslice clk) [2]	N/A	N/A	N/A	
Pixel Rate, blend, 32bpp (pix/clk, memory, @ 1.0x unslice clk) [2]				
Pixel Rate, blend, 32bpp (pix/clk, memory, @ 1.5x unslice clk) [2]	N/A	N/A	N/A	
Media Attributes				
Samplers (media)	2	3	3	6
VDBox Instances	1	1	1	2
VEBox Instances	1	1	1	2
SFC Instances	1	1	1	1

Product Configuration Attribute Table				
Display Attributes				
Display Pipes	3	3	3	3
Display Planes per Pipe	3	3	3	3
DDI ports	2	2	2	2
eDP ports	1	1	1	1

Footnotes:

* Architectural Name = Slice Count x Subslice Count x EUs per Subslice

[a] SKU naming & details has not yet been decided.

[b] One EU reserved for die recovery purposes.

Steppings and Device IDs

The following table lists currently proposed variations of GT Die / Packages for Gen9 (KBL). Prior to manufacturing, this information is subject to change at any time.

CPU SKU	GT SKU	CPU Stepping	GT/Disp Stepping	Device0 ID	Native Device2 ID	GT Device2 Revision ID	Comments
KBL-U 2+2	GT2	H0	C0/B0	0x5904	0x5916	0x2	MB
KBL-Y 2+2	GT2	H0	C0/B0	0x590C	0x591E	0x2	MB
KBL-H 4+2	GT2	B0	F0/C0	0x5910	0x591B	0x4	DT, uP Server
KBL-S 4+2	GT2	B0	F0/C0	0x591F	0x5912	0x4	DT, uP Server
KBL-U 2+3	GT3	J1	D1/B1	0x5904	0x5926	0x6	
KBL-S 2+2	GT2	S0	F0/C0	0x590F	0x5912	0x4	

Kaby Lake SKUs and Device IDs

The following table details all currently planned SKUs for KBL. This list is subject to change at any time based on roadmap plans.

SKU Type	CPU SKU	GT SKU	Device 2 ID
U	KBL U - ULT 2+1F	GT1	0x5906
U	KBL U - ULT 2+3E, 28W	GT3	0x5927
U	KBL U - ULT 2+3E, 15W	GT3	0x5926
Y	KBL Y - ULX 2+2	GT2	0x591E
S	KBL DT 4+2	GT2	0x5912
S	KBL DT 2+2	GT2	0x5912
S	KBL DT 2+1F	GT1	0x5902
H	KBL Halo 4+2	GT2	0x591B
SRV	KBL SRV 4+2	GT2	0x591A
WS	KBL WKS 4+2	GT2	0x591D

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 5: Memory Views

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Introduction **1**
Memory Views Glossary 1
GPU Memory Interface **1**
Global Arbitration **2**
GFX MMIO – MCHBAR Aperture **2**
Graphics Memory Interface Functions **3**
Graphics Memory Clients 3
Graphics Memory Addressing Overview 4
Graphics Address Path 4
Graphics Memory Paths 5
Graphics Memory Address Spaces 6
Address Tiling Function Introduction **7**
Linear vs Tiled Storage 8
Auxiliary Surfaces For Sampled Tiled Resources 11
HiZ 11
CCS 11
MCS 12
Tile Formats 12
Tile-X Legacy Format 12
Tile-Y Legacy Format 13
W-Major Tile Format 14
Tile-Yf Format 15
Tile-Ys Format 16
Tiling Algorithm 17
Tiled Channel Select Decision 27
Tiling Support 27
Tiled (Fenced) Regions 28
Tiled Surface Parameters 28
Tiled Surface Restrictions 29
Per-Stream Tile Format Support 31
Main Memory 32

Optimizing Main Memory Allocation 32
Application of the Theory (Page Coloring) 33
3D Color and Depth Buffers 33
Media/Video 34
Physical Graphics Address Types 34
Graphics Virtual Memory 34
Graphics Translation Tables 35
GFX Page Tables 35
Tiled Resources Translation Tables 35
Registers for TR-TT Management 38
Detection and Treatment of Null and Invalid Tiles 41
TR-TT Modes 42
Virtual Addressed TR Translation Tables 42
TR-TT Page Walk 44
Page Table Modes 45
Per Process GTT 45
Page Tables Entry (PTE) Formats 45
Pointer to PML4 Table 50
PML4E: Pointer to PDP Table 51
PDPE: Pointer to PD Table 52
PD: Pointer to Page Table 54
PTE: Page Table Entry for 64KB Page 57
PTE: Page Table Entry for 4KB Page 58
PPGTT for 32b Virtual Address 60
Walk with 64KB Page 63
Walk with 2MB Page 64
Walk with 1GB Page 64
PPGTT for Standard Context (64b VA) 65
Walk with 64KB Page 71
Walk with 2MB Page 72
Walk with 1GB Page 74
Global GTT 75
Page Table Entry 75

Page Walk 76
Legacy mode with 32b VA 77
Page Walk in Legacy mode with 32b VA 77
Walk with 64KB Page 79
Page Table Entry (PTE) Formats 80
PDE for Page Table 80
PTE: Page Table Entry for 64KB Page 81
PTE: Page Table Entry for 4KB Page 82
Legacy mode with 48b VA 83
Page Walk in Legacy 48b Mode 83
Walk with 64KB Page 84
Walk with 2MB Page 86
Walk with 1GB Page 87
Page Tables Entry PTE Formats 88
Pointer to PML4 table 89
PML4E: Pointer to PDP Table 89
PDPE: Pointer to PD Table 89
PDPE for PD 90
PDPE for 1GB Page 91
PD: Pointer to Page Table 92
PDE for Page Table 92
PDE for 2MB Page 93
PTE: Page Table Entry for 64KB Page 94
PTE: Page Table Entry for 4KB Page 95
Advanced mode with 48b VA and IA32e Support 96
Page Walk in Advanced Mode 96
Walk with 2MB Page 98
Walk with 1GB Page 99
Page Tables Entry (PTE) Formats 100
Pointer to PML4 table 100
PML4E: Pointer to PDP Table 101
PDPE: Pointer to PD Table 102
PDPE for PD 102

PDPE for 1GB Page 104
PD: Pointer to Page Table 105
PDE for Page Table 105
PDE for 2MB Page 107
PTE: Page Table Entry for 64KB Page 108
PTE: Page Table Entry for 4KB Page 110
GTT Cache 111
GFX Page Walker (GAM) 111
Context Definition for GFX Page Walker 112
Context Definition Delivery 114
Element Descriptor Register 115
PDP0/PML4/PASID Descriptor Register 116
PDP1 Descriptor Register 117
PDP2 Descriptor Register 117
PDP3 Descriptor Register 117
List of Registers and Command Streamers 118
Updating Page Table Pointers (aka PD Load) 120
Page Walker (GAM) Reset 121
TLB Caching and Management 122
TLB Caches 124
Context Cache - CC 124
PASID Cache - PC 124
Intermediate Page Walk Caches (PML4, PDP, PD) – PWC 124
TLB – Final Page Entry 124
TLB Entry Content 125
TLB Accessed and Dirty Flags 127
Updating A/D Bits 128
Replacement 133
Invalidations of TLB 133
Optional Invalidations 133
GTT Walk Request Port (HDC) 134
TLB Invalidation 138
Shared Virtual Memory 139

Page Table Types 140
Page Faults 141
SVM Coherency 142
Page Faults 143
Hardware and Software Pieces for Page Faults 144
Page Access Rights 145
Page Fault Decision 147
Fault Aware Surfaces 147
Page Fault Modes 148
Fault and Hang/Crash (Legacy Mode) 149
Fault and Halt 150
Fault Flows and Impact 151
Client/GAM Interface 151
Rescheduling a Faulted Context 153
Application Termination 154
Memory Types and Cache Interface 155
Memory Object Control State (MOCS) 155
MOCS Registers 156
L3 Control Registers 156
Memory Interface Control Registers 158
HDCL3GAM Change Specific to Coherent L3 162
Graphics Cache and Memory Interface 163
Skip Caching in LLC and eDRAM 163
Caching Display Surfaces in LLC 164
Page Walker Access and Memory Types 165
Page Walker Memory Types 165
Error Cases 165
Common Surface Formats 166
Non-Video Surface Formats 166
Surface Format Naming 166
Intensity Formats 166
Luminance Formats 166
R1_UNORM (same as R1_UINT) and MONO8 167

Palette Formats 168
P4A4_UNORM 168
A4P4_UNORM 168
P8A8_UNORM 169
A8P8_UNORM 169
P8_UNORM 170
P2_UNORM 170
Compressed Surface Formats 171
ETC1_RGB8 171
ETC2_RGB8 and ETC2_SRGB8 173
T mode 174
H mode 176
Planar mode 178
EAC_R11 and EAC_SIGNED_R11 179
ETC2_RGB8_PTA and ETC2_SRGB8_PTA 181
Differential Mode 181
T and H Modes 181
Planar Mode 182
ETC2_EAC_RGBA8 and ETC2_EAC_SRGB8_A8 182
EAC_RG11 and EAC_SIGNED_RG11 182
FXT Texture Formats 184
Overview of FXT1 Formats 184
FXT1 CC_HI Format 184
CC_HI Block Encoding 185
CC_HI Block Decoding 185
FXT1 CC_CHROMA Format 186
CC_CHROMA Block Encoding 186
CC_CHROMA Block Decoding 187
FXT1 CC_MIXED Format 188
CC_MIXED Block Encoding 188
CC_MIXED Block Decoding 189
FXT1 CC_ALPHA Format 193
CC_ALPHA Block Encoding 193

CC_ALPHA Block Decoding 193
DXT/BC1-3 Texture Formats 195
Opaque and One-bit Alpha Textures (DXT1/BC1) 196
Opaque Textures (DXT1_RGB) 198
Compressed Textures with Alpha Channels (DXT2-5 / BC2-3) 199
BC4 201
BC5 202
BC6H 204
Field Definition 204
Endpoint Computation 216
Palette Color Computation 216
Texel Selection 217
ONE Mode 218
TWO Mode 218
BC7 219
Field Definition 219
Endpoint Computation 225
Palette Color Computation 225
Texel Selection 226
ONE Mode 226
TWO Mode 226
THREE Mode 228
Adaptive Scalable Texture Compression (ASTC) 230
ASTC Fundamentals 231
Background 231
New Surface Formats for ASTC Texture 234
ASTC File Format and Memory Layout 239
ASTC Header Data Structure and Amendment 239
Data Layout in ASTC Compression File 240
Total ASTC Data Block Layout in All Mipmap Levels 240
Data Layout in Memory for All Mipmap Levels 241
ASTC Data Structure 244
Layout and Description of Block Data 244

Partitioning 245
Index Mode 245
Index Planes 249
Index Infill Procedure 249
Color Endpoint Mode 250
Color Endpoint Data Size Determination 252
Void-Extent Blocks 253
Decoding Process 254
Overview Decoding Flow 254
Integer Sequence Encoding 257
Endpoint Unquantization 258
LDR Endpoint Decoding 259
HDR Endpoint Decoding 262
HDR Endpoint Mode 2 (HDR Luminance, Large Range) 263
HDR Endpoint Mode 3 (HDR Luminance, Small Range) 263
HDR Endpoint Mode 7 (HDR RGB, Base+Scale) 264
HDR Endpoint Mode 11 (HDR RGB, Direct) 266
HDR Endpoint Mode 14 (HDR RGB, Direct + LDR Alpha) 270
HDR Endpoint Mode 15 (HDR RGB, Direct + HDR Alpha) 270
Restrictions on Number of Partitions Per Block 271
Index Decoding 272
Index Unquantization 272
Infill Process 273
Index Application 275
Dual-Plane Decoding 276
Partition Pattern Generation 276
Data Size Determination 278
3D Void-Extent Blocks 279
Illegal Encodings 280
Profile Support 280
Video Pixel/Texel Formats 281
Packed Memory Organization 281
Planar Memory Organization 282

Raw Format 285
Surface Memory Organizations 285
Display, Overlay, Cursor Surfaces 285
2D Render Surfaces 285
2D Monochrome Source 286
2D Color Pattern 286
3D Color Buffer (Destination) Surfaces 286
3D Depth Buffer Surfaces 287
3D Separate Stencil Buffer Surfaces 287
Surface Layout and Tiling 288
Maximum Surface Size in Bytes 288
Tiling 288
Typed Buffers 289
MIP Layout 290
Raw (Untyped) Buffers 292
Structured Buffers 292
1D Surfaces 292
Tiling and Mip Tail for 1D Surfaces 293
1D Alignment Requirements 294
2D Surfaces 295
Calculating Texel Location 296
Tiling and Mip Tails for 2D Surfaces 298
2D/CUBE Alignment Requirement 301
Multisampled 2D Surfaces 301
Interleaved Multisampled Surfaces 302
Compressed Multisampled Surfaces 302
Uncompressed Multisampled Surfaces 304
Quilted Textures 305
Cube Surfaces 305
3D Surfaces 307
Tiling and Mip Tails for 3D Surfaces 309
3D Alignment Requirements 311
Surface Padding Requirements 312

Alignment Unit Size 312
Alignment Parameters 312
Sampling Engine Surfaces 313
Render Target and Media Surfaces 314
Device2 PASID Capability Structures 314
PASID Extended Capability 314
PASID Extended Capability Header 315
PASID Capability Register 316
PASID Control Register 317
ATS Extended Capability 318
ATS Extended Capability Header 318
ATS Capability Register 318
ATS Control Register 318
Page Request Extended Capability 319
Page Request Extended Capability Header 319
Page Request Control Register 320
Page Request Status Register 321
Outstanding Page Request Capacity 322
Outstanding Page Request Allocation 322
Atomics for Page Table Updates (MSQD) 323
Implementation 326
Atomic_Page_update_0000: 326
Atomic_Page_update_0001: 327
Atomic_Page_update_0010: 327
Atomic_Page_update_0011: 328
Atomic_Page_update_0100: 328
Atomic_Page_update_0101: 329
Atomic_Page_update_0110: 330
Atomic_Page_update_0111: 331
Atomic_Page_update_1000: 332
Atomic_Page_update_1001: 333
Atomic_Page_update_1010: 334
Atomic_Page_update_1011: 335

Atomic_Page_update_1100: 336
Atomic_Page_update_1101: 337
Atomic_Page_update_1110: 338
Atomic_Page_update_1101: 339
Atomic_A_update_000: 340
Atomic_A_update_001: 341
Atomic_A_update_010: 342
Atomic_A_update_011: 343
Atomic_A_update_100: 344
Atomic_AD_update_101: 345
Atomic_AD_update_110: 346
Atomic_AD_update_111: 347
Atomic Operations between GPU and IA 348

Introduction

The hardware supports three engines:

- The Render command streamer interfaces to 3D/IE and display streams.
- The Media command streamer interfaces to the fixed function media.
- The Blitter command streamer interfaces to the blit commands.

Software interfaces of all three engines are very similar and should only differ on engine-specific functionality.

Memory Views Glossary

Term	Definition
IOMMU	I/O Memory Mapping unit
SVM	Shared Virtual Memory, implies the same virtual memory view between the IA cores and processor graphics.
Page Walker (GAM)	GFX page walker which handles page level translations between GFX virtual memory to physical memory domain.

GPU Memory Interface

GPU memory interface functions are divided into 4 different major sections:

- Global Arbitration
- Memory Interface Functions
- Page Translations (GFX Page Walker)
- Ring Interface Functions (GTI)

GT Interface functions are covered at a different chapter/HAS and not part of this documentation. The following documentation is meant for GFX arbitration paths in accessing to memory/cache interfaces and page translations and page walker functions.

Global Arbitration

The global memory arbitration fabric is meant to be a hierarchal memory fabric where memory accesses from different stages of the pipeline are consolidated to a single interface towards GT's connection to CPU's ring interface.

The arbitration on the fabric is programmable via a simple per pipeline stage priority levels.

Programming Note	
Context:	Global Memory Arbitration
Arbitration allows 4 levels of arbitration where each pipeline level can be put into these 4 levels. Each consolidation stage simply follows the 4-level arbitration with grace periods to allow ahead of the pipeline to get a higher share of the memory bandwidth.	

The final arbitration takes places in GAM between parallel compute engines. Each engine (in some cases major pipeline stages are also separated, i.e. Z vs Color vs L3 vs Fixed Functions) gets a count in a grace period where its accesses are counted against a global pool. If a particular engine (or pipeline stage) exhausts its max allowed, it is dropped to a lower priority and goes to fixed pipeline based prioritization. Once all counts are expired, the grace period completes and resets.

The count values are programmable via MMIO (i.e. *_MAX_REQ_COUNT) registers with defaults favoring the pipeline order.

GFX MMIO – MCHBAR Aperture

Address: 140000h – 147FFFh

Default Value: Same as MCHBAR

Access: Aligned Word, Dword, or Qword Read/Write

This range defined in the graphics MMIO range is an alias with which graphics driver can read and write registers defined in the MCHBAR MMIO space claimed through Device #0. Attributes for registers defined within the MCHBAR space are preserved when the same registers are accessed via this space. Registers that the graphics driver requires access to are Rank Throttling, GMCH Throttling, Thermal Sensor, etc.

The Alias functions work for MMIO access from the CPU only. A command stream load register immediate will drop the data, and the store register immediate will return all Zeroes.

Graphics MMIO registers can be accessed through MMIO BARs in function #0 and function #1 in Device #2. The aliasing mechanism is turned off if memory access to the corresponding function is turned off via software or in certain power states.

Please refer to applicable EDS documentation of offset 140000h – 147FFFh for details of this register's format and behavior.

Graphics Memory Interface Functions

The major role of an integrated graphics device's Memory Interface (MI) function is to provide various client functions access to "graphics" memory used to store commands, surfaces, and other information used by the graphics device. This chapter describes the basic mechanisms and paths by which graphics memory is accessed.

Information not presented in this chapter includes:

- Microarchitectural and implementation-dependent features (e.g., internal buffering, caching, and arbitration policies).
- MI functions and paths specific to the operation of external (discrete) devices attached via external connections.
- MI functions essentially unrelated to the operation of the internal graphics devices, .e.g., traditional "chipset functions"
- GFX Page Walker and GT interface functions are covered in different chapters.

Graphics Memory Clients

The MI function provides memory access functionality to a number of external and internal graphics memory *clients*, as described in the table below.

Graphics Memory Clients

MI Client	Access Modes
Host Processor	Read/Write of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses
External PEG Graphics Device	**Write-Only** of Graphics Operands located in Main Memory via the Graphics Aperture. (This client is not described in this chapter).
Peer PCI Device	**Write-Only** of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses (i.e., mapped by GTT*). Note that DMI access to Graphics registers is not supported.*
Coherent Read/Write (internal)	Internally-generated snooped reads/writes.
Command Stream (internal)	DMA Read of graphics commands and related graphics data.
Vertex Stream (internal)	DMA Read of indexed vertex data from Vertex Buffers by the 3D Vertex Fetch (VF) Fixed Function.
Instruction/State Cache (internal)	Read of pipelined 3D rendering state used by the 3D/Media Functions and instructions executed by the EUs.
Render Cache (internal)	Read/Write of graphics data operated upon by the graphics rendering engines (Blt, 3D, MPEG, etc.) Read of render surface state.
Sampler Cache (internal)	Read of texture (and other sampled surface) data stored in graphics memory.

MI Client	Access Modes
Display/Overlay Engines (internal)	Read of display, overlay, cursor and VGA data.
Media Engines	Read and write of media content and media processing.
uController	Read/Write (DMA) functions for u-controller and scheduler.

Graphics Memory Addressing Overview

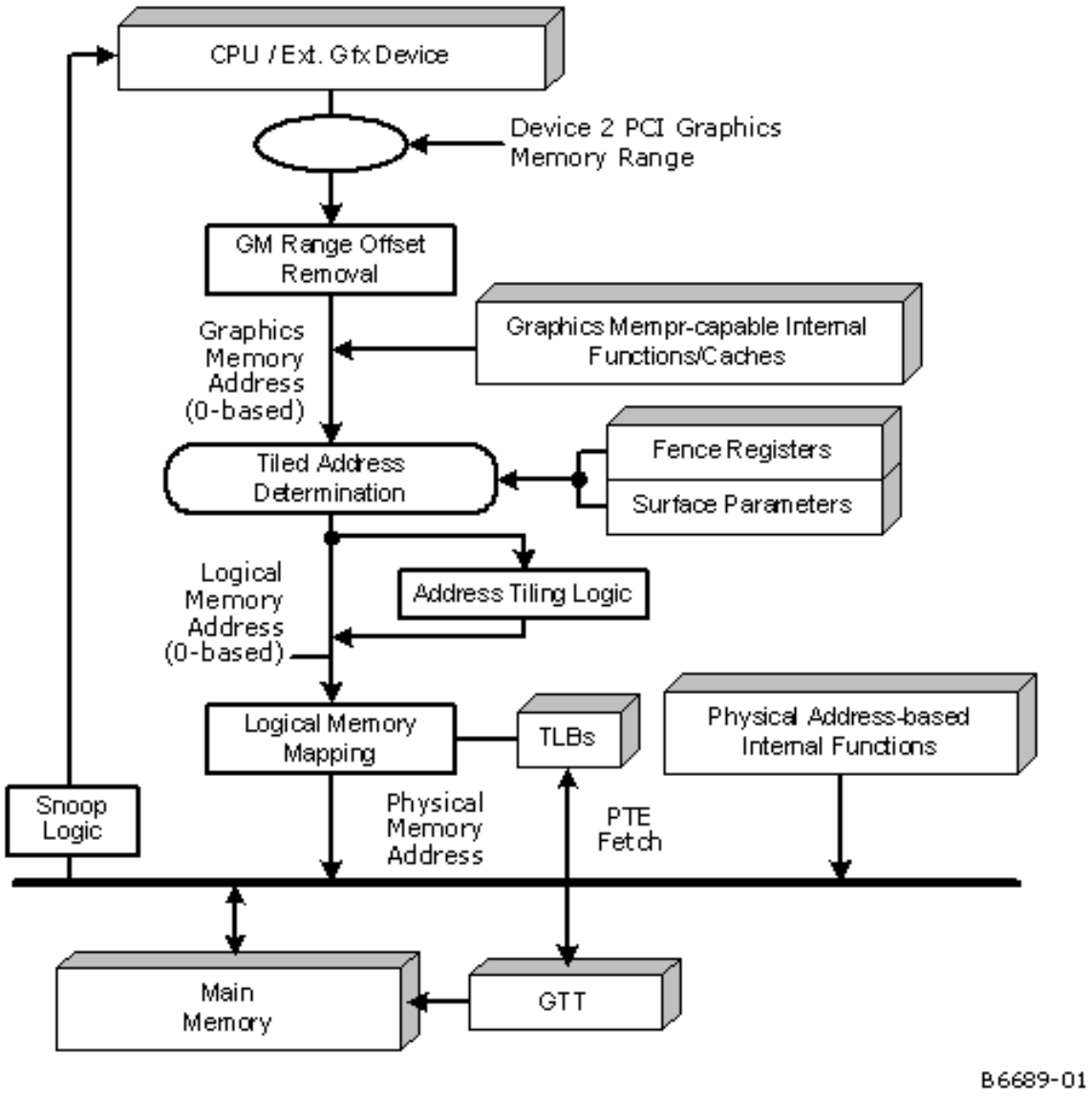
The Memory Interface function provides access to graphics memory (GM) clients. It accepts memory addresses of various types, performs a number of optional operations along *address paths*, and eventually performs reads and writes of graphics memory data using the resultant addresses. The remainder of this subsection will provide an overview of the graphics memory clients and address operations.

Graphics Address Path

Graphics Address Path shows the internal graphics memory address path, connection points, and optional operations performed on addresses. Externally-supplied addresses are normalized to zero-based *Graphics Memory* (*GM) addresses* (GM_Address). If the GM address is determined to be a tiled address (based on inclusion in a fenced region or via explicit surface parameters), *address tiling* is performed. At this point the address is considered a *Logical Memory address*, and is translated into a *Physical Memory address* via the GTT and associated TLBs. The physical memory location is then accessed.

CPU accesses to graphics memory are sent back on the ring to snoop. Hence pages that are mapped cacheable in the GTT will be coherent with the CPU cache if accessed through graphics memory aperture.

Graphics Memory Paths



The remainder of this chapter describes the basic features of the graphics memory address pipeline, namely Address Tiling, Logical Address Mapping, and Physical Memory types and allocation considerations.

Graphics Memory Address Spaces

The *Graphics Memory Address Spaces* table lists the five supported Graphics Memory Address Spaces. Note that the Graphics Memory Range Removal function is automatically performed to transform system addresses to internal, zero-based Graphics Addresses.

Graphics Memory Address Types

Address Type	Description	Range	Gen9
GMADR	Address range allocated via the Device 2 (integrated graphics device) GMADR register. The processor and other peer (DMI) devices utilize this address space to read/write graphics data that resides in Main Memory. This address is internally converted to a GM_Address.	This is a 4 GB BAR above physical memory.	128 MB, 256 MB, 512 MB, 1GB, 2GB, 4GB
GTTMMADR	The combined Graphics Translation Table Modification Range and Memory Mapped Range. The range requires 16 MB combined for MMIO and Global GTT aperture, with 8MB of that used by MMIO and 8MB used by GTT. GTTADR will begin at GTTMMADR 8MB while the MMIO base address will be the same as GTTMMADR. For the Global GTT, this range is defined as a memory BAR in graphics device config space. It is an alias into which software is required to write Page Table Entry values PTEs. Software may read PTE values from the global Graphics Translation Table GTT. PTEs cannot be written directly into the global GTT memory area.	This is a 16MB BAR above physical memory.	16 MB (2 MB MMIO + 6 MB reserved + 8 MB GGTT)
GSM	GTT Stolen Memory. It is an 8 MB (max) region taken out of physical memory to store the Global GTT entries for page translations specific to GFX driver use. It is accessible via GTTMMADR from the CPU path however GPU/DE can access the same region directly.	This is an 8 MB region in physical memory not visible to OS.	1 MB, 2 MB, 4 MB, 8 MB
DSM	Data stolen memory, the size is determined with GMS filed (8 bits) with MAX size of 4 GB. This is a stolen memory which can be accessed via GMADR for CPU and directly for GPU/DE. Size is programmable with 32 MB multiplier. First 4KB of DSM has to be reserved for GFX hardware use.	This is a max of 4 GB stolen physical memory for GFX data structures.	0 MB, 32 MB, 64 MB, 96 MB, ...4096MB

Next level breakdown for GTTMMADR is given below.

Software is allowed to use range x17_0000 to x17_FFFF as the Null range.

Address Tiling Function Introduction

When dealing with memory operands (e.g., graphics surfaces) that are inherently rectangular in nature, certain functions within the graphics device support the storage/access of the operands using alternative (tiled) memory formats to increase performance. This section describes these memory storage formats, why and when they should be used, and the behavioral mechanisms within the device to support them.

Legacy Tiling Modes:

- **TileY**: Used for most tiled surfaces when **TR_MODE**=TR_NONE.
- **TileX** : Used primarily for display surfaces.
- **TileW:** Used for Stencil surfaces.

Programming Note	
Context:	Address Tiling Function

Tiled Resource Tiling Modes

- **TileYF:** 4KB tiling mode based on TileY
- **TileYS:** 64KB tiling mode based on TileY

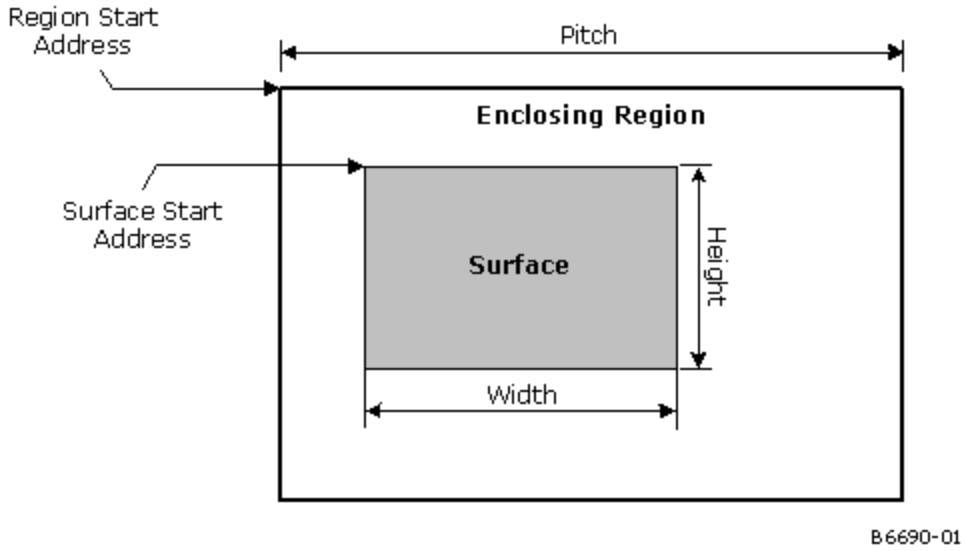
These modes are based on 4KB and 64KB tiles. The 64KB tile is made up of a 4x4 matrix of 4KB tiles. The 4KB tiles in general have a different layout as compared to the legacy modes, with the sub-mode defining the layout within the 4KB tile. The sub-modes are determined by the bits per element of the surface format. The Tiled Resource Mode field in SURFACE_STATE is used to select the new modes.

Tiled surface base addresses must be tile aligned (64KB aligned for TileYS, 4KB aligned for all other tile modes). For 1D surfaces, the base address must be 64KB aligned if **Tiled Resource Mode** is TRMODE_64KB, and 4KB aligned if **Tiled Resource Mode** is TRMODE_4KB. An exception to this tile alignment is when a SURFACE_STATE describes a single MIP within the MIP Tail of another surface, using a 64-bit or 128-bit **Surface Format**—then **Surface Base Address** can refer directly to the given MIP (e.g. to write to a non-renderable **Surface Format** by re-describing as an alternative surface).

Linear vs Tiled Storage

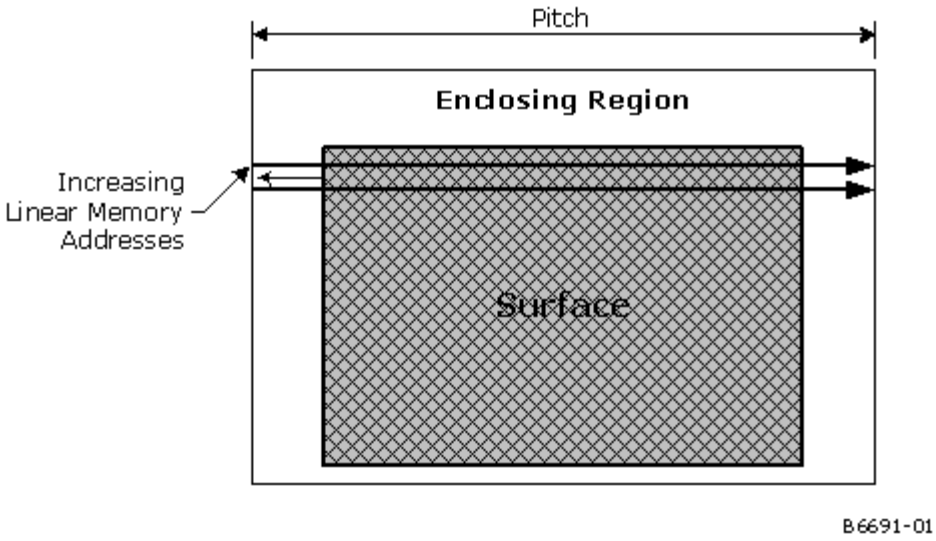
Regardless of the memory storage format, "rectangular" memory operands have a specific *width* and *height*, and are considered as residing within an enclosing rectangular region whose width is considered the *pitch* of the region and surfaces contained within. Surfaces stored within an enclosing region must have widths less than or equal to the region pitch (indeed the enclosing region may coincide exactly with the surface). *Rectangular Memory Operand Parameters* shows these parameters.

Rectangular Memory Operand Parameters



The simplest storage format is the *linear* format (see *Linear Surface Layout*), where each row of the operand is stored in sequentially increasing memory locations. If the surface width is less than the enclosing region's pitch, there will be additional memory storage between rows to accommodate the region's pitch. The pitch of the enclosing region determines the distance (in the memory address space) between vertically-adjacent operand elements (e.g., pixels, texels).

Linear Surface Layout

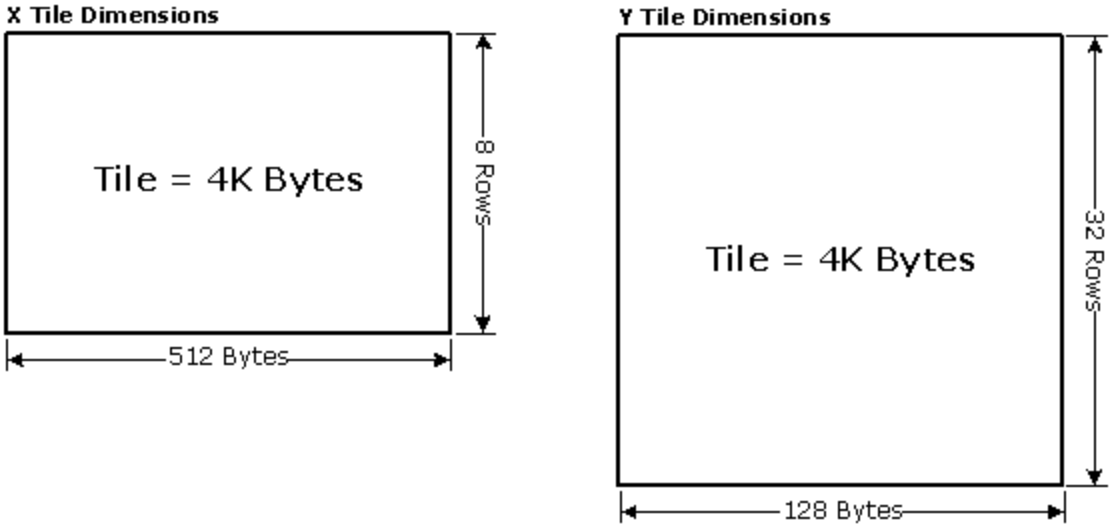


The linear format is best suited for 1-dimensional row-sequential access patterns (e.g., a display surface where each scanline is read sequentially). Here the fact that one object element may reside in a different memory page than its vertically-adjacent neighbors is not significant; all that matters is that horizontally-adjacent elements are stored contiguously. However, when a device function needs to access a 2D subregion within an operand (e.g., a read or write of a 4x4 pixel span by the 3D renderer, a read of a 2x2 texel block for bilinear filtering), having vertically-adjacent elements fall within different memory pages is to be avoided, as the page crossings required to complete the access typically incur increased memory latencies (and therefore lower performance).

One solution to this problem is to divide the enclosing region into an array of smaller rectangular regions, called memory *tiles*. Surface elements falling within a given tile will all be stored in the same physical memory page, thus eliminating page-crossing penalties for 2D subregion accesses within a tile and thereby increasing performance.

Tiles have a fixed 4KB size and are aligned to physical DRAM page boundaries. They are either 8 rows high by 512 bytes wide or 32 rows high by 128 bytes wide (see *Memory Tile Dimensions*). Note that the dimensions of tiles are irrespective of the data contained within – e.g., a tile can hold twice as many 16-bit pixels (256 pixels/row x 8 rows = 2K pixels) than 32-bit pixels (128 pixels/row x 8 rows = 1K pixels).

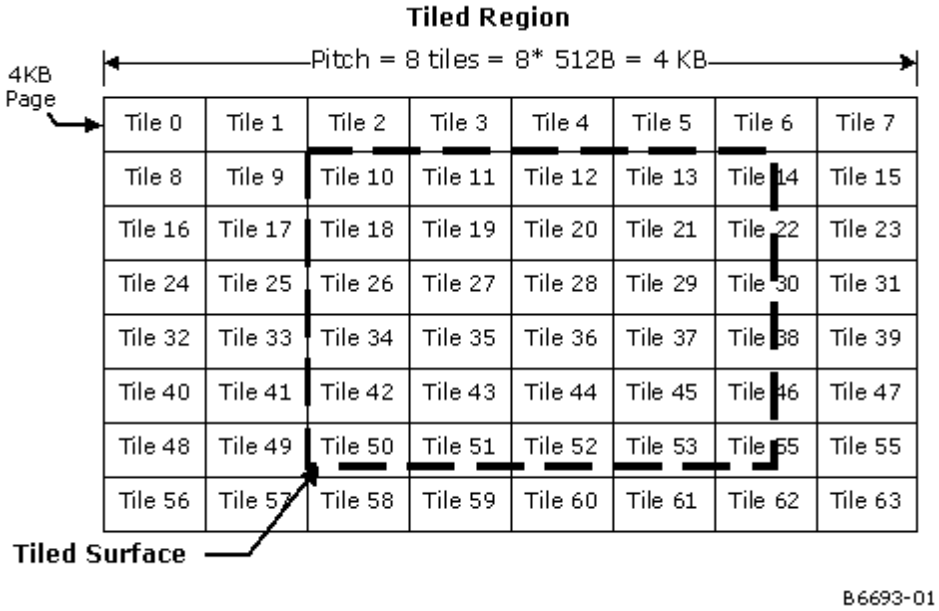
Memory Tile Dimensions



The pitch of a tiled enclosing region must be an integral number of tile widths. The 4KB tiles within a tiled region are stored sequentially in memory in row-major order.

The *Tiled Surface Layout* figure shows an example of a tiled surface located within a tiled region with a pitch of 8 tile widths (512 bytes * 8 = 4KB). Note that it is the *enclosing region* that is divided into tiles – the surface is not necessarily aligned or dimensioned to tile boundaries.

Tiled Surface Layout



Auxiliary Surfaces For Sampled Tiled Resources

For surfaces which are defined as Tiled Resources (TileYs or TileYf format), there may be auxiliary surfaces which are associated with the surface (e.g. HiZ, CCS or MCS). These auxiliary surfaces, while actually not defined as TileYs or TileYf will behave like tiled resources from the hardware perspective. It is possible for software to map and unmap tiles of auxiliary surfaces as tiles of the associated surface are mapped and unmapped. Below is a description how sampling to the mapped/unmapped tile resources is handled for the associated auxiliary surface. Normally, sampling unmapped tiles will return a NULL response to the requesting agent.

HiZ

A tile of HiZ data must be mapped to memory whenever any depth surface (Z) pixels associated with the HiZ tile are mapped. When all Z pixels associated with a HiZ tile are unmapped, the HiZ tile may be mapped or unmapped. Below is a table showing the responses for sampling to mapped and unmapped depth surfaces.

Responses for Sampling to A Depth-Surface Tiled Resource

Depth Surface Mapping	HiZ Surface Mapping	Sample Response
Mapped	Mapped	Normal Sample Response
Mapped	Unmapped	Undefined
Unmapped	Mapped	NULL Response
Unmapped	Unmapped	NULL Response

A "NULL Response" means that the sample returned will be all 0's and the **Null Pixel Mask** (if requested) will indicate the depth pixel is Null.

CCS

A tile of CCS (Color Control Surface) must be mapped to memory whenever color surface pixels associated with the CCS tile are mapped. When all color pixels associated with a CCS tile are unmapped, the CCS may be mapped or unmapped. CCS is used to indicate that the color surface is losslessly compressed. Below is a table showing the responses for sampling to mapped and unmapped.

Responses for Sampling to a Losslessly Compressed Color Surface That is a Tiled Resource

Color Surface Mapping	CCS Surface Mapping	Sample Response
Mapped	Mapped	Normal Response
Mapped	Unmapped	Undefined
Unmapped	Mapped	NULL Response
Unmapped	Unmapped	NULL Response

A "NULL Response" means that the sample returned will be all 0's and the **Null Pixel Mask** (if requested) will indicate the depth pixel is Null.

MCS

A tile of MCS(Multi-Sample Control Surface) must be mapped to memory whenever MSAA surface pixels associated with the CCS tile are mapped. When all MSAA pixels associated with a MCS tile are unmapped, the MCS may be mapped or unmapped. Below is a table showing the responses for sampling to mapped and unmapped.

Responses for Sampling to MSAA Tiled Resources

MSAA Surface Mapping	MCS Mapping	Sample Response
Mapped	Mapped	Normal Response
Mapped	Unmapped	Undefined Response
Unmapped	Mapped	NULL Response
Unmapped	Unmapped	NULL Response

A "NULL Response" means that the sample returned will be all 0's and the **Null Pixel Mask** (if requested) will indicate the depth pixel is Null.

Tile Formats

Multiple tile formats are supported by the Gen Core. The following sections define and describe these formats.

Tiling formats are controlled by programming the fields Tile_Mode and Tiled_Resource_Mode in the RENDER_SURFACE_STATE.

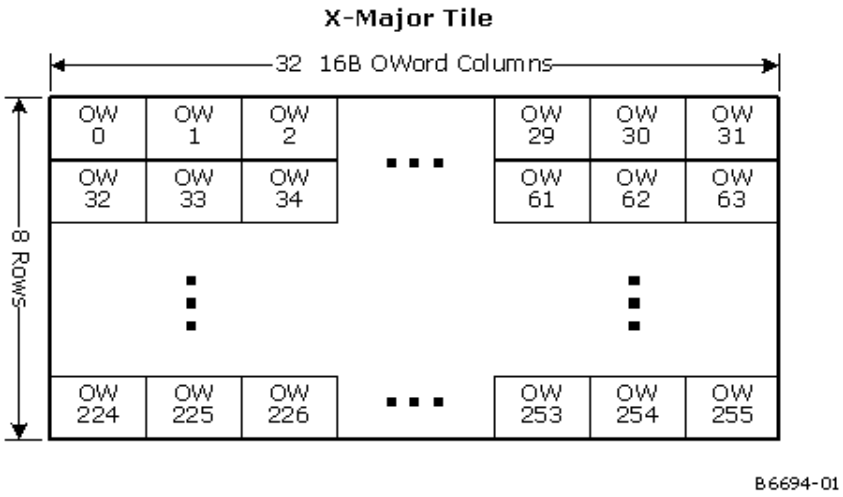
Tile-X Legacy Format

The legacy format Tile-X is a *X-Major* (row-major) storage of tile data units, as shown in the following figure. It is a 4KB tile which is subdivided into an 8-high by 32-wide array of 16-byte OWords . The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles. Note that an X-major tiled region with a tile pitch of 1 tile is actually stored in a linear fashion.

Tile-X format is selected for a surface by programming the Tiled_Mode field in RENDER_SURFACE_STATE to XMAJOR.

For 3D sampling operation, a surface using Tile-X layout is generally lower performance the organization of texels in memory.

Tile X-Tile (X-Major) Layout



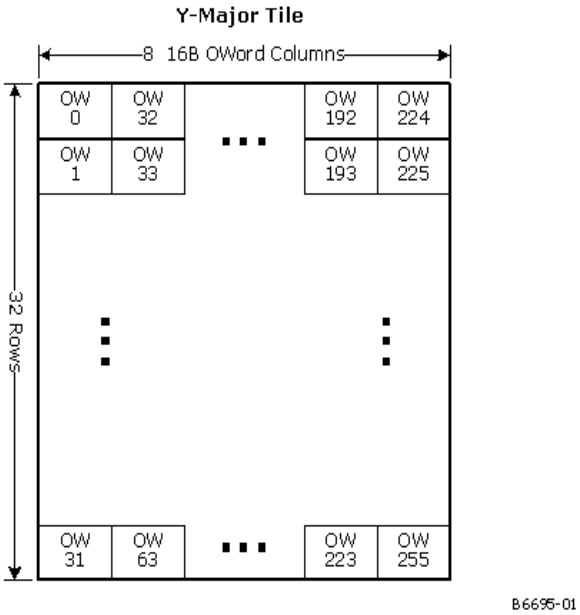
Tile-Y Legacy Format

The device supports Tile-Y legacy format which is *Y-Major* (column major) storage of tile data units, as shown in the following figure. A 4KB tile is subdivided 32-high by 8-wide array of OWords. The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles.

Tile-Y surface format is selected by programming the Tile_Mode field in RENDER_SURFACE_STATE to YMAJOR.

Note that 3D sampling of a surface in Tile-Y format is usually has higher performance due to the layout of pixels.

Y-Major Tile Layout

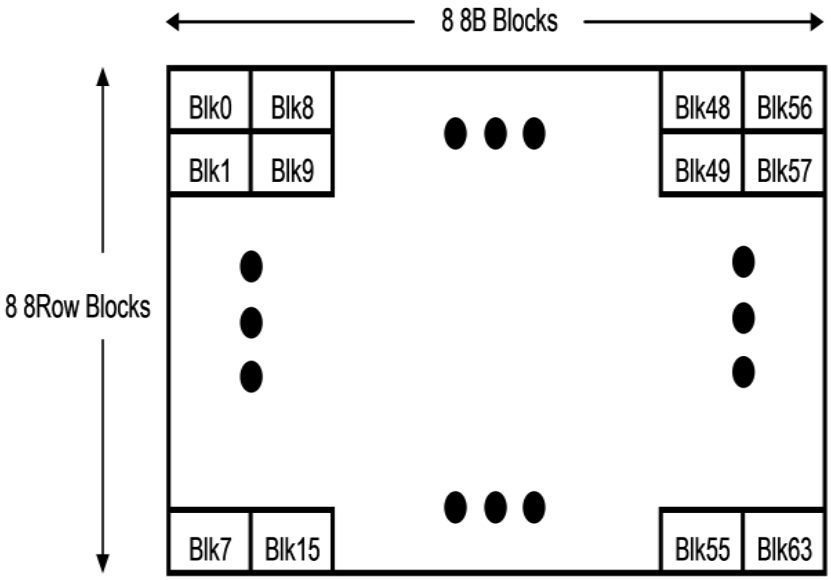


W-Major Tile Format

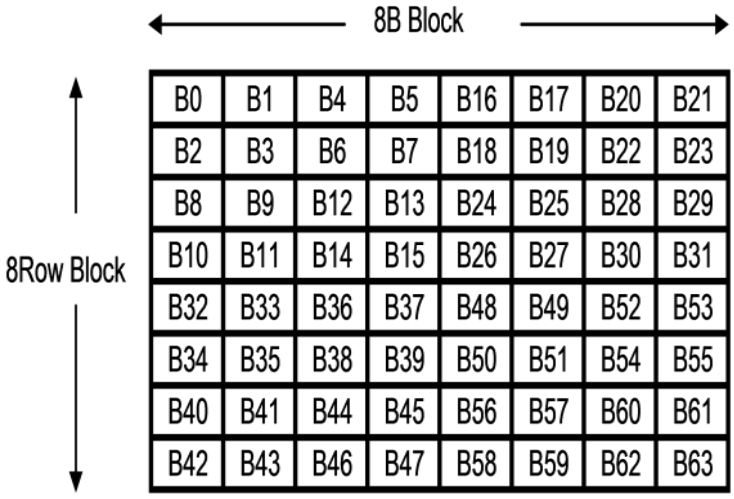
The device supports additional format *W-Major* storage of tile data units, as shown in the following figures. A 4KB tile is subdivided into 8-high by 8-wide array of Blocks for W-Major Tiles (W Tiles). Each Block is 8 rows by 8 bytes. The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles. W-Major Tile Format is used for separate stencil.

Tile-W surface format is selected by programming the Tile_Mode field in the RENDER_SURFACE_STATE to WMAJOR.

W-Major Tile Layout



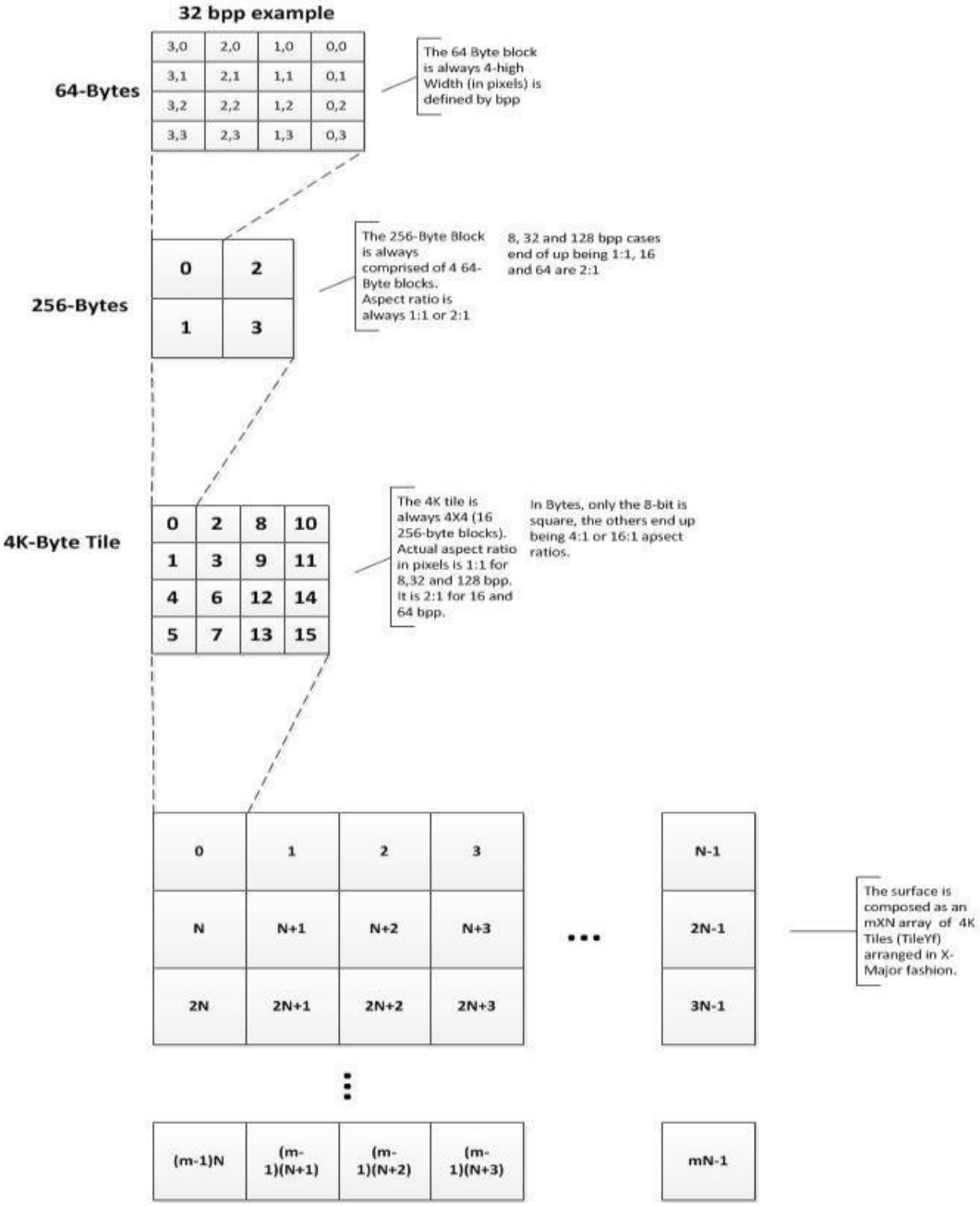
W-Major Block Layout



Tile-Yf Format

Tile-Yf is a 4K-Byte tile format (similar to Tile-Y), but organized in a different manner. Tile-Yf is selected by programming the Tile_Mode field in the RENDER_SURFACE_STATE to YMAJOR and the Tiled_Resource_Mode to TILEYF. The diagram below shows how pixels are mapped into the TileYf format for 2D surfaces, and it uses 32Bpp (bits per pixel) surface format as an example on a 2D surface which is N tiles wide and m tiles high. The exact aspect ratio will be dependent on the Bpp of the surface. Note that the TileYf format is identical to the TileYs up to the 4K-Byte tile size.

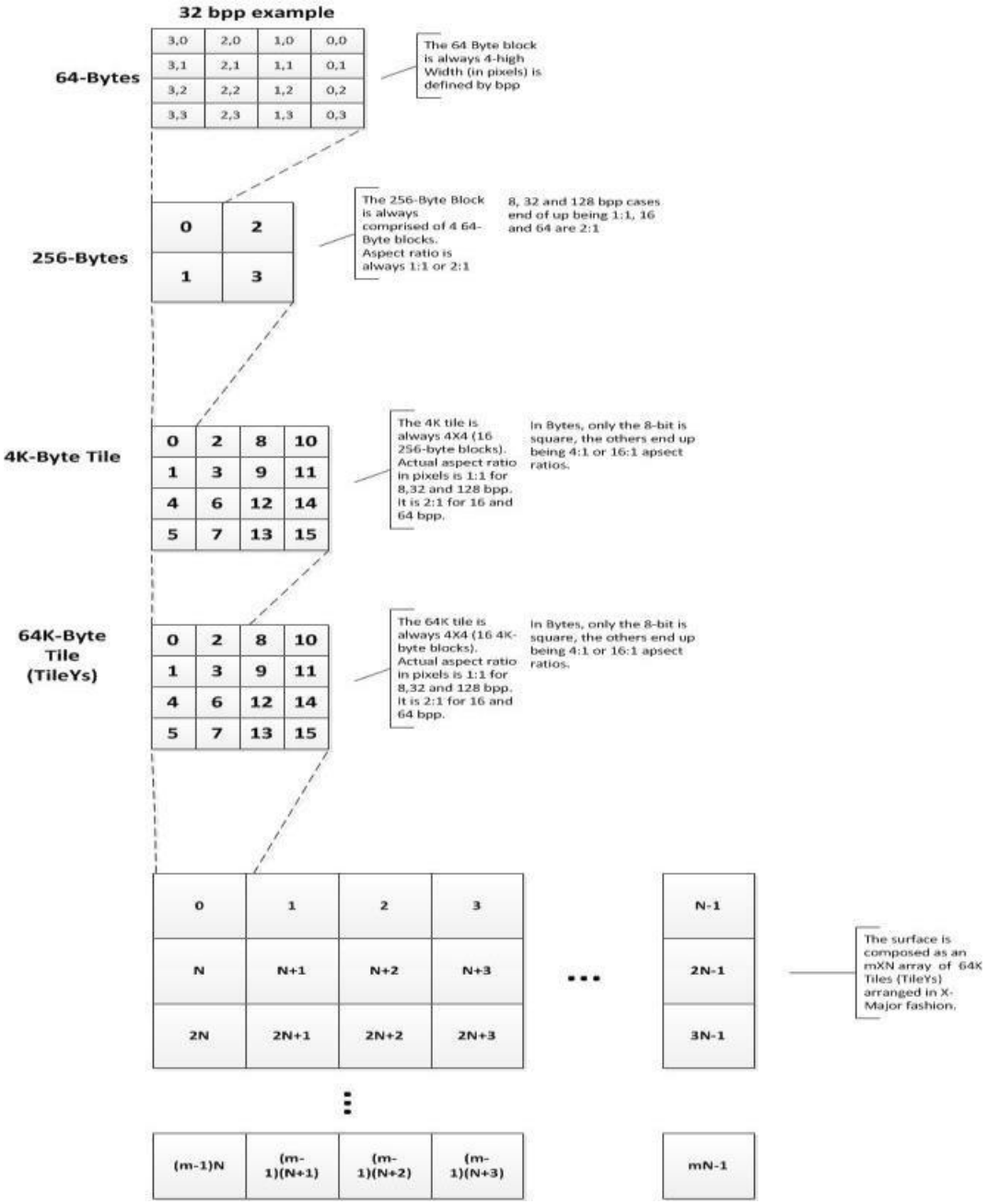
2D Tile Layout for TileYf



Tile-Ys Format

TileYs is a 64K-Byte tile size. It is enabled by programming the Tile_Mode field (in RENDER_SURFACE_STATE) to YMAJOR, and programming the Tiled_Resource_Mode to TILEYS. It is organized as shown below, and is composed of 4KByte blocks which have identical layout to the TileYf format. The diagram below shows how pixels are mapped into the TileYs format, and it uses 32Bpp (bits per pixel) surface format as an example on a 2D surface which is N tiles wide and m tiles high. The exact aspect ratio will be dependent on the Bpp of the surface.

Tile-Ys Layout



Tiling Algorithm

The following pseudo-code describes the algorithm for translating a tiled memory surface in graphics memory to an address in logical space.

The following new modes are supported for Tiled Resources (**TR_MODE** != TR_NONE) defined to enable tiled resources.

For more details about Mip Tails, see Surface Layout and Tiling in the Common Surface Formats section.

- **TileYF:** 4KB tiling mode based on TileY
- **TileYS:** 64KB tiling mode based on TileY

```
Inputs:
      LinearAddress(offset into regular or LT aperture in terms of bytes),
     Pitch(in terms of tiles),
             WalkY (1 for Y and 0 for X)
             WalkW (1 for W and 0 for the rest)

Static Parameters:
      TileH (Height of tile, 8 for X, 32 for Y and 64 for W),
      TileW (Width of Tile in bytes, 512 for X, 128 for Y and 64 for W)
      TileSize = TileH * TileW;
      RowSize = Pitch * TileSize;

If (Fenced) {
     LinearAddress = LinearAddress – FenceBaseAddress
     LinearAddrInTileW = LinearAddress div TileW;
     Xoffset_inTile = LinearAddress mod TileW;
       Y = LinearAddrInTileW div Pitch;
       X = LinearAddrInTileW mod Pitch + Xoffset_inTile;
}

// Internal graphics clients that access tiled memory already have the X, Y
// coordinates and can start here
YOff_Within_Tile = Y mod TileH;
XOff_Within_Tile = X mod TileW;
TileNumber_InY = Y div TileH;
TileNumber_InX = X div TileW;

      TiledOffsetY = RowSize * TileNumber_InY + TileSize * TileNumber_InX + TileH * 16 *
(XOff_Within_Tile div 16) +
             YOff_Within_Tile * 16 +
             (XOff_Within_Tile mod 16);

      TiledOffsetW = RowSize * TileNumber_InY +
             TileSize * TileNumber_InX +
             TileH * 8 * (XOff_Within_Tile div 8) +
             64 * (YOff_Within_Tile div 8) +
             32 * ((YOff_Within_Tile div 4) mod 2) +
             16 * ((XOff_Within_Tile div 4) mod 2) +
              8 * ((YOff_Within_Tile div 2) mod 2) +
              4 * ((XOff_Within_Tile div 2) mod 2) +
              2 * (YOff_Within_Tile mod 2) +
             (XOff_Within_Tile mod 2);

     TiledOffsetX = RowSize * TileNumber_InY + TileSize * TileNumber_InX + TileW *
YOff_Within_Tile + XOff_Within_Tile;
```

```
      TiledOffset = WalkW? TiledOffsetW : (WalkY? TiledOffsetY : TiledOffsetX);

        TiledAddress = Tiled? (BaseAddress + TiledOffset): (BaseAddress + Y*LinearPitch +
X);TiledAddress = (Tiled &&
              (Address Swizzling for Tiled-Surfaces == 01)) ?
              (WalkW || WalkY) ?
              (TiledAddress div 128) * 128 +
              (((TiledAddress div 64) mod 2) ^
              ((TiledAddress div 512) mod 2)) +
              (TiledAddress mod 32)
              :
              (TiledAddress div 128) * 128 +
              (((TiledAddress div 64) mod 2) ^
              ((TiledAddress div 512) mod 2)
              ((TiledAddress Div 1024) mod2) +
              (TiledAddress mod 32)
              :
              TiledAddress;
       }
```

Address Swizzling for Tiled-Surfaces is no longer used because the main memory controller has a more effective address swizzling algorithm.

For Address Swizzling for Tiled-Surfaces see ARB_MODE – Arbiter Mode Control register, ARB_CTL—Display Arbitration Control 1 and TILECTL - Tile Control register

The Y-Major tile formats have the characteristic that a surface element in an even row is located in the same aligned 64-byte cacheline as the surface element immediately below it (in the odd row). This spatial locality can be exploited to increase performance when reading 2x2 texel squares for bilinear texture filtering, or reading and writing aligned 4x4 pixel spans from the 3D Render pipeline.

On the other hand, the X-Major tile format has the characteristic that horizontally-adjacent elements are stored in sequential memory addresses. This spatial locality is advantageous when the surface is scanned in row-major order for operations like display refresh. For this reason, the Display and Overlay memory streams only support linear or X-Major tiled surfaces (Y-Major tiling is not supported by these functions). This has the side effect that 2D- or 3D-rendered surfaces must be stored in linear or X-Major tiled formats if they are to be displayed. Non-displayed surfaces, e.g., “rendered textures”, can also be stored in Y-Major order.

The following Psuedo Code Describes the algorithm for mapping TileYs and TileYf Tile Address to Byte Offset within a Tile. It describes the support for 2D for both TileYs and TileYf as well as MSAA 2D For TileYs.

```
/*****************************************************************************\
   BitMask
     Used for masking single bits of x, y, z, ss# when _pdep32 instruction is
    not available
\*****************************************************************************/
enum BitMask
{
    BIT0 = 1,
    BIT1 = (1 « 1),
    BIT2 = (1 « 2),
    BIT3 = (1 « 3),
    BIT4 = (1 « 4),
    BIT5 = (1 « 5),
    BIT6 = (1 « 6),
    BIT7 = (1 « 7),
    BIT8 = (1 « 8),
    BIT9 = (1 « 9),
    BIT10 = (1 « 10),
    BIT11 = (1 « 11),
    BIT12 = (1 « 12),
    BIT13 = (1 « 13),
    BIT14 = (1 « 14),
    BIT15 = (1 « 15)
};
/*****************************************************************************\
    TileYS/TileYF constant swizzle masks w/o _pdep32 instruction

    Used to mask contiguous x/y/z/sample bit groupings before being shifted into
    their final swizzled bit positions
\*****************************************************************************/
// used for fallback 'manual' bit shifting
static const UINT16 xMaskBits5_4  = 0x0030;
static const UINT16 xMaskBits3_0  = 0x000F;
static const UINT16 yMaskBits4_0  = 0x001F;
static const UINT16 yMaskBits3_0  = 0x000F;
static const UINT16 yMaskBits2_0  = 0x0007;
static const UINT16 yMaskBits1_0  = 0x0003;
static const UINT16 SampleMask3_0 = 0x000F;
static const UINT16 SampleMask2_0 = 0x0007;
static const UINT16 SampleMask1_0 = 0x0003;
static const UINT16 SampleMask0   = 0x0001;

/*****************************************************************************\
    TileYS 2D Tile address swizzling functions w/o _pdep32
\*****************************************************************************/
/*
 ____________________________________________________________________________
|    Num    | Bits per element |           Tiled element offset bits           |
|  Samples  |                  |15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|___________|__________________|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|     1x    |     64 & 128     |x9|y5|x8|y4|x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|           |     16 & 32      |x8|y6|x7|y5|x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|           |        8         |x7|y7|x6|y6|x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYS2dElementOffset64_128bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYS swizzled bit locations
    xSwizzle = ((BIT9 & x) « 6) |
               ((BIT8 & x) « 5) |
```

```
                ((BIT7 & x) « 4) |
                ((BIT6 & x) « 3) |
                ((xMaskBits5_4 & x) « 2) | // shift to bit positions 7..6
                (xMaskBits3_0 & x);

    ySwizzle = ((BIT5 & y) « 9) |
                ((BIT4 & y) « 8) |
                ((BIT3 & y) « 7) |
                ((BIT2 & y) « 6) |
                ((yMaskBits1_0 & y) « 4);   // shift to bit positions 5..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}

UINT16 TileYS2dElementOffset16_32bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYS swizzled bit locations
    xSwizzle = ((BIT8 & x) « 7) |
                ((BIT7 & x) « 6) |
                ((BIT6 & x) « 5) |
                ((BIT5 & x) « 4) |
                ((BIT4 & x) « 3) |
                (xMaskBits3_0 & x);

    ySwizzle = ((BIT6 & y) « 8) |
                ((BIT5 & y) « 7) |
                ((BIT4 & y) « 6) |
                ((BIT3 & y) « 5) |
                ((yMaskBits2_0 & y) « 4); // shift to bit positions 6..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}

UINT16 TileYS2dElementOffset8bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYS swizzled bit locations
    xSwizzle = ((BIT7 & x) « 8) |
                ((BIT6 & x) « 7) |
                ((BIT5 & x) « 6) |
                ((BIT4 & x) « 5) |
                (xMaskBits3_0 & x);

    ySwizzle = ((BIT7 & y) « 7) |
                ((BIT6 & y) « 6) |
                ((BIT5 & y) « 5) |
                ((yMaskBits4_0 & y) « 4); // shift to bit positions 8..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}

/*******************************************************************************\
    TileYS 2D MSAA Tile address swizzling functions w/o _pdep32
\*******************************************************************************/
```

```
/*
 _____________________________________________________________________________
|    Num     | Bits per element |           Tiled element offset bits            |
|  Samples   |                  |15 |14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|____________________________|___|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|     2x     |     64 & 128     |ss0|y5|x8|y4|x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|            |     16 & 32      |ss0|y6|x7|y5|x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|            |        8         |ss0|y7|x6|y6|x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYS2xMsaaElementOffset64_128bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT8 & x) « 5) |                  // shift to bit position 13
               ((BIT7 & x) « 4) |                  // shift to bit position 11
               ((BIT6 & x) « 3) |                  // shift to bit position 9
               ((xMaskBits5_4 & x) « 2) |          // shift to bit positions 7..6
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT5 & y) « 9) |                  // shift to bit position 14
               ((BIT4 & y) « 8) |                  // shift to bit position 12
               ((BIT3 & y) « 7) |                  // shift to bit position 10
               ((BIT2 & y) « 6) |                  // shift to bit position 8
               ((yMaskBits1_0 & y) « 4);           // shift to bit positions 5..4

    SampleSwizzle = (sample && SampleMask0) « 15;  // shift to bit position 15

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS2xMsaaElementOffset16_32bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT7 & x) « 6) |                  // shift to bit position 13
               ((BIT6 & x) « 7) |                  // shift to bit position 11
               ((BIT5 & x) « 6) |                  // shift to bit position 9
               ((BIT4 & x) « 5) |                  // shift to bit position 7
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT6 & y) « 8) |                  // shift to bit position 14
               ((BIT5 & y) « 7) |                  // shift to bit position 12
               ((BIT4 & y) « 6) |                  // shift to bit position 10
               ((BIT3 & y) « 5) |                  // shift to bit position 8
               ((yMaskBits2_0 & y) « 4);           // shift to bit positions 6..4

    SampleSwizzle = (sample && SampleMask0) « 15;  // shift to bit position 15

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS2xMsaaElementOffset8bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
```

```
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT6 & x) « 7) |                   // shift to bit position 13
               ((BIT5 & x) « 6) |                   // shift to bit position 11
               ((BIT4 & x) « 5) |                   // shift to bit position 9
               (xMaskBits3_0 & x);                   // leave in bits 3..0

    ySwizzle = ((BIT7 & y) « 7) |                   // shift to bit position 14
               ((BIT6 & y) « 6) |                   // shift to bit position 12
               ((BIT5 & y) « 5) |                   // shift to bit position 10
               ((yMaskBits4_0 & y) « 4);            // shift to bit positions 8..4

    SampleSwizzle = (sample && SampleMask0) « 15;  // shift to bit position 15

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

/*
 ____________________________________________________________________________
|    Num    | Bits per element |           Tiled element offset bits           |
|  Samples  |                  |15 |14 |13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|___________|__________________|___|___|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|     4x    |     64 & 128     |ss1|ss0|x8|y4|x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|           |     16 & 32      |ss1|ss0|x7|y5|x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|           |        8         |ss1|ss0|x6|y6|x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYS4xMsaaElementOffset64_128bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT8 & x) « 5) |                   // shift to bit position 13
               ((BIT7 & x) « 4) |                   // shift to bit position 11
               ((BIT6 & x) « 3) |                   // shift to bit position 9
               ((xMaskBits5_4 & x) « 2) |           // shift to bit positions 7..6
               (xMaskBits3_0 & x);                   // leave in bits 3..0

    ySwizzle = ((BIT4 & y) « 8) |                   // shift to bit position 12
               ((BIT3 & y) « 7) |                   // shift to bit position 10
               ((BIT2 & y) « 6) |                   // shift to bit position 8
               ((yMaskBits1_0 & y) « 4);            // shift to bit positions 5..4

    SampleSwizzle = (sample && SampleMask1_0) « 14;// shift to bit positions 15..14

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS4xMsaaElementOffset16_32bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT7 & x) « 6) |                   // shift to bit position 13
               ((BIT6 & x) « 7) |                   // shift to bit position 11
```

```
                ((BIT5 & x) « 6) |                  // shift to bit position 9
                ((BIT4 & x) « 5) |                  // shift to bit position 7
                (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT5 & y) « 7) |                   // shift to bit position 12
               ((BIT4 & y) « 6) |                   // shift to bit position 10
               ((BIT3 & y) « 5) |                   // shift to bit position 8
               ((yMaskBits2_0 & y) « 4);            // shift to bit positions 6..4

    SampleSwizzle = (sample && SampleMask1_0) « 14;// shift to bit positions 15..14

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS4xMsaaElementOffset8bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT6 & x) « 7) |                   // shift to bit position 13
               ((BIT5 & x) « 6) |                   // shift to bit position 11
               ((BIT4 & x) « 5) |                   // shift to bit position 9
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT6 & y) « 6) |                   // shift to bit position 12
               ((BIT5 & y) « 5) |                   // shift to bit position 10
               ((yMaskBits4_0 & y) « 4);            // shift to bit positions 8..4

    SampleSwizzle = (sample && SampleMask1_0) « 14;// shift to bit positions 15..14

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

/*
 ____________________________________________________________________________
|    Num     | Bits per element |           Tiled element offset bits          |
|  Samples   |                  |15 |14 |13 |12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|____________________________|___|___|___|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|     8x     |     64 & 128     |ss2|ss1|ss0|y4|x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|            |     16 & 32      |ss2|ss1|ss0|y5|x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|            |        8         |ss2|ss1|ss0|y6|x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYS8xMsaaElementOffset64_128bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT7 & x) « 4) |                   // shift to bit position 11
               ((BIT6 & x) « 3) |                   // shift to bit position 9
               ((xMaskBits5_4 & x) « 2) |           // shift to bit positions 7..6
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT4 & y) « 8) |                   // shift to bit position 12
               ((BIT3 & y) « 7) |                   // shift to bit position 10
               ((BIT2 & y) « 6) |                   // shift to bit position 8
               ((yMaskBits1_0 & y) « 4);            // shift to bit positions 5..4
```

```
    SampleSwizzle = (sample && SampleMask2_0) « 13;// shift to bit positions 15..13

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS8xMsaaElementOffset16_32bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT6 & x) « 7) |                  // shift to bit position 11
               ((BIT5 & x) « 6) |                  // shift to bit position 9
               ((BIT4 & x) « 5) |                  // shift to bit position 7
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT5 & y) « 7) |                  // shift to bit position 12
               ((BIT4 & y) « 6) |                  // shift to bit position 10
               ((BIT3 & y) « 5) |                  // shift to bit position 8
               ((yMaskBits2_0 & y) « 4);           // shift to bit positions 6..4

    SampleSwizzle = (sample && SampleMask2_0) « 13;// shift to bit positions 15..13

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS8xMsaaElementOffset8bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT5 & x) « 6) |                  // shift to bit position 11
               ((BIT4 & x) « 5) |                  // shift to bit position 9
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT6 & y) « 6) |                  // shift to bit position 12
               ((BIT5 & y) « 5) |                  // shift to bit position 10
               ((yMaskBits4_0 & y) « 4);           // shift to bit positions 8..4

    SampleSwizzle = (sample && SampleMask2_0) « 13;// shift to bit positions 15..13

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

/*
 ____________________________________________________________________________________________
|    Num    | Bits per element |           Tiled element offset bits                        |
|  Samples  |                  |15 |14 |13 |12 |11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|___________|__________________|___|___|___|___|__|__|__|__|__|__|__|__|__|__|__|__|
|    16x    |     64 & 128     |ss3|ss2|ss1|ss0|x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|           |     16 & 32      |ss3|ss2|ss1|ss0|x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|           |        8         |ss3|ss2|ss1|ss0|x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYS16xMsaaElementOffset64_128bpe(UINT16 x, UINT16 y, UINT16 sample)
{
```

```
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT7 & x) « 4) |                  // shift to bit position 11
               ((BIT6 & x) « 3) |                  // shift to bit position 9
               ((xMaskBits5_4 & x) « 2) |          // shift to bit positions 7..6
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT3 & y) « 7) |                  // shift to bit position 10
               ((BIT2 & y) « 6) |                  // shift to bit position 8
               ((yMaskBits1_0 & y) « 4);           // shift to bit positions 5..4

    SampleSwizzle = (sample && SampleMask3_0) « 12;// shift to bit positions 15..12

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS16xMsaaElementOffset16_32bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT6 & x) « 7) |                  // shift to bit position 11
               ((BIT5 & x) « 6) |                  // shift to bit position 9
               ((BIT4 & x) « 5) |                  // shift to bit position 7
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT4 & y) « 6) |                  // shift to bit position 10
               ((BIT3 & y) « 5) |                  // shift to bit position 8
               ((yMaskBits2_0 & y) « 4);           // shift to bit positions 6..4

    SampleSwizzle = (sample && SampleMask3_0) « 12;// shift to bit positions 15..12

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}

UINT16 TileYS16xMsaaElementOffset8bpe(UINT16 x, UINT16 y, UINT16 sample)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;
    UINT16 SampleSwizzle;

    // shift bits in x, y, and sample to their respective TileYS MSAA swizzled bit locations
    xSwizzle = ((BIT5 & x) « 6) |                  // shift to bit position 11
               ((BIT4 & x) « 5) |                  // shift to bit position 9
               (xMaskBits3_0 & x);                  // leave in bits 3..0

    ySwizzle = ((BIT5 & y) « 5) |                  // shift to bit position 10
               ((yMaskBits4_0 & y) « 4);           // shift to bit positions 8..4

    SampleSwizzle = (sample && SampleMask3_0) « 12;// shift to bit positions 15..12

    // OR the swizzled bit positions for final offset within a tile
    return SampleSwizzle | xSwizzle | ySwizzle;
}
```

```
/*****************************************************************************\
    TileYF 2D Tile address swizzling functions w/o _pdep32
\*****************************************************************************/
/*
 ____________________________________________________________________________
|    Num     | Bits per element |           Tiled element offset bits           |
|  Samples   |                  |15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
|____________|__________________|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|     1x     |     64 & 128     |  |  |  |  |x7|y3|x6|y2|x5|x4|y1|y0|x3|x2|x1|x0|
|            |     16 & 32      |  |  |  |  |x6|y4|x5|y3|x4|y2|y1|y0|x3|x2|x1|x0|
|            |        8         |  |  |  |  |x5|y5|x4|y4|y3|y2|y1|y0|x3|x2|x1|x0|
*/
UINT16 TileYF2dElementOffset64_128bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYF swizzled bit locations
    xSwizzle = ((BIT7 & x) « 4) |
               ((BIT6 & x) « 3) |
               ((xMaskBits5_4 & x) « 2) | // shift to bit positions 7..6
               (xMaskBits3_0 & x);

    ySwizzle = ((BIT3 & y) « 7) |
               ((BIT2 & y) « 6) |
               ((yMaskBits1_0 & y) « 4);   // shift to bit positions 5..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}

UINT16 TileYF2dElementOffset16_32bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYF swizzled bit locations
    xSwizzle = ((BIT6 & x) « 5) |
               ((BIT5 & x) « 4) |
               ((BIT4 & x) « 3) |
               (xMaskBits3_0 & x);

    ySwizzle = ((BIT4 & y) « 6) |
               ((BIT3 & y) « 5) |
               ((yMaskBits2_0 & y) « 4); // shift to bit positions 6..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}

UINT16 TileYF2dElementOffset8bpe(UINT16 x, UINT16 y)
{
    UINT16 xSwizzle;
    UINT16 ySwizzle;

    // shift bits in x and y to their respective TileYF swizzled bit locations
    xSwizzle = ((BIT5 & x) « 6) |
               ((BIT4 & x) « 5) |
               (xMaskBits3_0 & x);

    ySwizzle = ((BIT5 & y) « 5) |
```

```
                ((yMaskBits4_0 & y) « 4); // shift to bit positions 8..4

    // OR the swizzled bit positions for final offset within a tile
    return xSwizzle | ySwizzle;
}
```

Tiled Channel Select Decision

There was a historical configuration control field to swizzle address bit[6] for in X/Y tiling modes. This was set in three different places: TILECTL[1:0], ARB_MODE[5:4], and DISP_ARB_CTL[14:13].

The swizzle fields are all reserved, and the CPU's memory controller performs all address swizzling modifications.

Tiling Support

The rearrangement of the surface elements in memory must be accounted for in device functions operating upon tiled surfaces. (Note that not all device functions that access memory support tiled formats). This requires either the modification of an element's linear memory address or an alternate formula to convert an element's X,Y coordinates into a tiled memory address.

However, before tiled-address generation can take place, some mechanism must be used to determine whether the surface elements accessed fall in a linear or tiled region of memory, and if tiled, what the tile region pitch is, and whether the tiled region uses X-Major or Y-Major format. There are two mechanisms by which this detection takes place: (a) an implicit method by detecting that the pre-tiled (linear) address falls within a "fenced" tiled region, or (b) by an explicit specification of tiling parameters for surface operands (i.e., parameters included in surface-defining instructions).

The following table identifies the tiling-detection mechanisms that are supported by the various memory streams.

Access Path	Tiling-Detection Mechanisms Supported
Processor access through the Graphics Memory Aperture	Fenced Regions
3D Render (Color/Depth Buffer access)	Explicit Surface Parameters
Sampled Surfaces	Explicit Surface Parameters
Blt operands	Explicit Surface Parameters
Display and Overlay Surfaces	Explicit Surface Parameters

Tiled (Fenced) Regions

The only mechanism to support the access of surfaces in tiled format by the host or external graphics client is to place them within "fenced" tiled regions within Graphics Memory. A fenced region is a block of Graphics Memory specified using one of the sixteen FENCE device registers. (See *Memory Interface Registers* for details). Surfaces contained within a fenced region are considered tiled from an external access point of view. Note that fences cannot be used to untile surfaces in the PGM_Address space since external devices cannot access PGM_Address space. Even if these surfaces (or any surfaces accessed by an internal graphics client) fall within a region covered by an enabled fence register, that enable will be effectively masked during the internal graphics client access. Only the explicit surface parameters described in the next section can be used to tile surfaces being accessed by the internal graphics clients.

Tiled Surface Parameters

Internal device functions require explicit specification of surface tiling parameters via information passed in commands and state. This capability is provided to limit the reliance on the fixed number of fence regions.

The following table lists the surface tiling parameters that can be specified for 3D Render surfaces (Color Buffer, Depth Buffer, Textures, etc.) via SURFACE_STATE.

Surface Parameter	Description
Tiled Surface	If ENABLED, the surface is stored in a tiled format. If DISABLED, the surface is stored in a linear format.
Tile Walk	If Tiled Surface is ENABLED, this parameter specifies whether the tiled surface is stored in Y-Major or X-Major tile format.
Base Address	Additional restrictions apply to the base address of a Tiled Surface vs. that of a linear surface.
Pitch	Pitch of the surface. Note that, if the surface is tiled, this pitch must be a multiple of the tile width.

Tiled Surface Restrictions

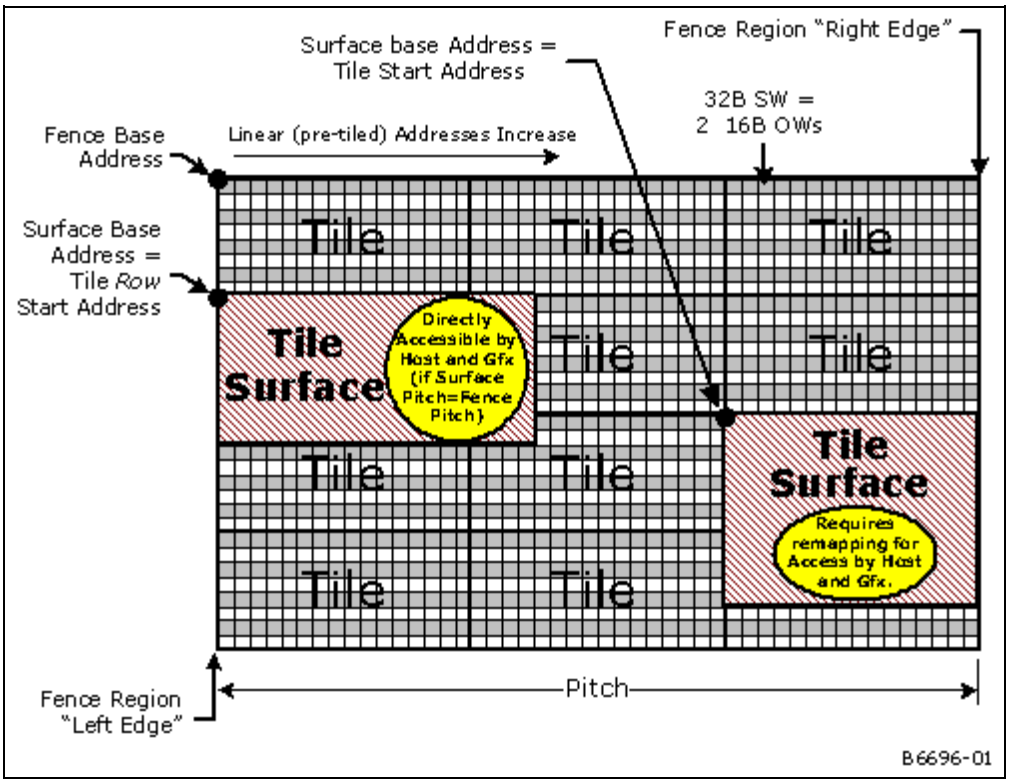
Additional restrictions apply to the Base Address and Pitch of a surface that is tiled. In addition, restrictions for tiling via SURFACE_STATE are subtly different from those for tiling via fence regions. The most restricted surfaces are those that will be accessed both by the host (via fence) and by internal device functions. An example of such a surface is a tiled texture that is initialized by the CPU and then sampled by the device.

The tiling algorithm for internal device functions is different from that of fence regions. Internal device functions always specify tiling in terms of a surface. The surface must have a base address, and this base address is not subject to the tiling algorithm. Only *offsets* from the base address (as calculated by X, Y addressing within the surface) are transformed through tiling. The base address of the surface must therefore be 4KB-aligned. This forces the 4KB tiles of the tiling algorithm to exactly align with 4KB device pages once the tiling algorithm has been applied to the offset. The width of a surface must be less than or equal to the surface pitch. There are additional considerations for surfaces that are also accessed by the host (via a fence region).

Fence regions have no base address per se. Host linear addresses that fall in a fence region are translated in their entirety by the tiling algorithm. It is as if the surface being tiled by the fence region has a base address in graphics memory equal to the fence base address, and all accesses of the surfaces are (possibly quite large) offsets from the fence base address. Fence regions have a virtual “left edge” aligned with the fence base address, and a “right edge” that results from adding the fence pitch to the “left edge”. Surfaces in the fence region must not straddle these boundaries.

Base addresses of surfaces that are to be accessed both by an internal graphics client and by the host have the tightest restrictions. In order for the surface to be accessed without GTT re-mapping, the surface base address (as set in SURFACE_STATE) must be a “Tile Row Start Address” (TRSA). The first address in each tile row of the fence region is a Tile Row Start Address. The first TRSA is the fence base address. Each TRSA can be generated by adding an integral multiple of the row size to the fence base address. The row size is simply the fence pitch in tiles multiplied by 4KB (the size of a tile.)

Tiled Surface Placement



The pitch in SURFACE_STATE must be set equal to the pitch of the fence that will be used by the host to access the surface if the same GTT mapping will be used for each access. If the pitches differ, a different GTT mapping must be used to eliminate the "extra" tiles (4KB memory pages) that exist in the excess rows at the right side of the larger pitch. Obviously no part of the surface that will be accessed can lie in pages that exist only in one mapping but not the other. The new GTT mapping can be done manually by SW between the time the host writes the surface and the device reads it, or it can be accomplished by arranging for the client to use a different GTT than the host (the PPGTT -- see *Logical Memory Mapping* below).

The width of the surface (as set in SURFACE_STATE) must be less than or equal to both the surface pitch and the fence pitch in any scenario where a surface will be accessed by both the host and an internal graphics client. Changing the GTT mapping will not help if this restriction is violated.

Surface Access	Base Address	Pitch	Width	Tile "Walk"
Host only	No restriction	Integral multiple of tile size <= 256KB	Must be <= Fence Pitch	No restriction
Client only	4KB-aligned	Integral multiple of tile size <= 256KB	Must be <= Surface Pitch	Restrictions imposed by the client (see Per Stream Tile Format Support)
Host and Client, No GTT Remapping	Must be TRSA	Fence Pitch = Surface Pitch = integral multiple of tile size <= 256KB	Width <= Pitch	Surface Walk must meet client restriction, Fence Walk = Surface Walk
Host and Client, GTT Remapping	4KB-aligned for client (will be tile-aligned for host)	Both must be Integral multiple of tile size <=128KB, but not necessarily the same	Width <= Min(Surface Pitch, Fence Pitch)	Surface Walk must meet client restriction, Fence Walk = Surface Walk

Per-Stream Tile Format Support

<table>
<tr><th>MI Client</th><th>Tile Formats Supported</th></tr>
<tr><td>CPU Read/Write</td><td>All</td></tr>
<tr><td>Display/Overlay</td><td>Y-Major not supported
X-Major required for Async Flips</td></tr>
<tr><td>Blt</td><td>Linear and X-Major only
No Y-Major support</td></tr>
<tr><td>3D Sampler</td><td>All Combinations of TileY, TileX and Linear are supported. TileY is the fastest, Linear is the slowest.</td></tr>
<tr><td>3D Color,Depth</td><td>
<table>
<tr><th>Rendering Mode
Color-vs-Depth bpp</th><th>Buffer Tiling Supported</th></tr>
<tr><td>Classical
Same Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
<tr><td>Classical
Mixed Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
</table>
</td></tr>
</table>

Main Memory

The integrated graphics device is capable of using 4KB pages of physical main (system) memory for graphics functions. Some of this main memory can be “stolen” from the top of system memory during initialization (e.g., for a VGA buffer). However, most graphics operands are dynamically allocated to satisfy application demands. To this end the graphics driver will frequently need to allocate locked-down (i.e., non-swappable) physical system memory pages – typically from a cacheable non-paged pool. The locked pages required to back large surfaces are typically non-contiguous. Therefore a means to support “logically-contiguous” surfaces backed by discontiguous physical pages is required. The Graphics Translation Table (GTT) that was described in previous sections provides the means.

Optimizing Main Memory Allocation

This section includes information for software developers on how to allocate SDRAM Main Memory (SM) for optimal performance in certain configurations. The general idea is that these memories are divided into some number of page types, and careful arrangement of page types both within and between surfaces (e.g., between color and depth surfaces) will result in fewer page crossings and therefore yield somewhat higher performance.

The algorithm for allocating physical SDRAM Main Memory pages to logical graphics surfaces is somewhat complicated by (1) permutations of memory device technologies (which determine page sizes and therefore the number of pages per device row), (2) memory device row population options, and (3) limitations on the allocation of physical memory (as imposed by the OS).

However, the theory to optimize allocation by limiting page crossing penalties is simple: (a) switching between open pages is optimal (again, the pages do not need to be sequential), (b) switching between memory device rows does not in itself incur a penalty, and (c) switching between pages within a particular bank of a row incurs a page miss and should therefore be avoided.

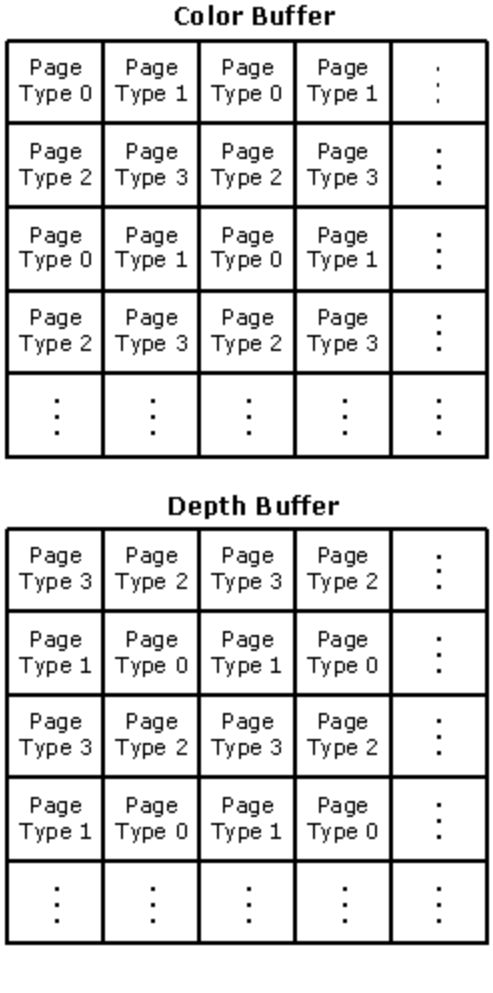
Application of the Theory (Page Coloring)

This section provides some scenarios for how Main Memory page allocation can be optimized.

3D Color and Depth Buffers

Here we want to minimize the impact of page crossings (a) between corresponding pages (1-4 tiles) in the Color and Depth buffers, and (b) when moving from a page to a neighboring page within a Color or Depth buffer. Therefore corresponding pages in the Color and Depth Buffers, and adjacent pages within a Color or Depth Buffer should be mapped to different page types (where a page's "type" or "color" refers to the row and bank it's in).

Memory Pages Backing Color and Depth Buffers



For higher performance, the Color and Depth Buffers could be allocated from *different* memory device rows.

Media/Video

The Y surfaces can be allocated using 4 page types in a similar fashion to the Color Buffer diagram. The U and V surfaces would split the same 4 page types as used in the Y surface.

Physical Graphics Address Types

The Physical Memory Address Types table lists the various *physical* address types supported by the integrated graphics device. Physical Graphics Addresses are either generated by Logical Memory mappings or are directly specified by graphics device functions. *These physical addresses are not subject to tiling or GTT re-mappings*.

Physical Memory Address Types

Address Type	Description	Range
MM_Address	Main Memory Address. Offset into physical, *unsnooped* Main Memory.	[0,TopOfMemory-1]
SM_Address	System Memory Address. Accesses are snooped in processor cache, allowing shared graphics/ processor access to (locked) cacheable memory data.	[0,512GB]

Graphics Virtual Memory

The GPU uses a virtual memory address space, where the graphics virtual address is mapped through a Page Table (PPGTT) to a physical memory address. Normally, this mapping is set up by the graphics device driver and is private to the GPU context. However, in some cases the graphics virtual address is shared with the CPU – see Shared Virtual Memory (SVM) for more information.

The range of valid graphics virtual addresses, and the types of page tables supported for address translation, varies with the GPU configuration. See the Configurations section for a summary the ranges and features supported by a specific graphics device.

Although the range of supported graphics virtual addresses varies, most GPU commands and GPU instructions use a common 64 bit definition for a graphics virtual address. Addresses outside of the supported range are reserved for future address space expansion. See the **GraphicsAddress** structure definition for specific details.

Some GPU devices support an extended graphics virtual memory address mapping called Tiled Resources. When enabled, the Tiled Resources Translation Table (TR-TT) pre-processes graphics virtual addresses. TR-TT maps a graphics virtual memory address either to a new graphics virtual memory address or to a Null Tile. Null Tiles return zero on reads and drop writes. For translations that are not Null Tiles, the new graphics virtual memory address is then used for the graphics virtual address and translated through the normal Page Table to generate a physical memory address.

Graphics Translation Tables

GT supports standard virtual memory models as defined by the IA programmer's guide. This section describes the different paging models, their behaviors, and the page table formats.

The Graphics Translation Tables GTT (Graphics Translation Table, sometimes known as the global GTT) and PPGTT (Per-Process Graphics Translation Table) are memory-resident page tables containing an array of DWord Page Translation Entries (PTEs) used in mapping logical Graphics Memory addresses to physical memory addresses, and sometimes snooped system memory "PCI" addresses.

The base address (MM offset) of the GTT and the PPGTT are programmed via the PGTBL_CTL and PGTBL_CTL2 MI registers, respectively. The translation table base addresses must be 4KB aligned. The GTT size can be either 128KB, 256KB, or 512KB (mapping to 128MB, 256MB, and 512MB aperture sizes respectively) and is physically contiguous. The global GTT should only be programmed via the range defined by GTTMMADR. The PPGTT is programmed directly in memory. The per-process GTT (PPGTT) size is controlled by the PGTBL_CTL2 register. The PPGTT can, in addition to the above sizes, also be 64KB in size (corresponding to a 64MB aperture). Refer to the GTT Range chapter for a bit definition of the PTE entries.

GFX Page Tables

GPU supports three page table mechanisms

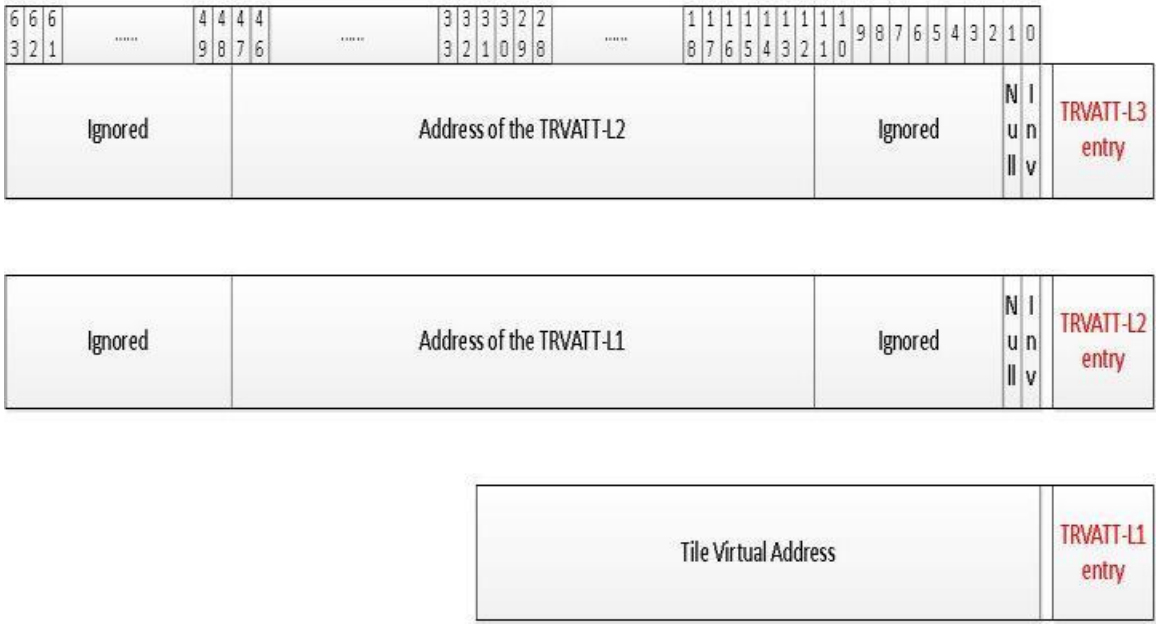
- IA32e compatible GTT
- PPGTT – private per process GTT (private GFX)
- GGTT - global GTT

All page tables use the same 64-bit PTE format. Differences are in how various bit fields applies (vs reserved) under various usage models.

Tiled Resources Translation Tables

Sparse Tiled Resources can be thought of as a kind of application-controlled virtual memory scheme. The application allocates a resource in a virtual address space. Then the application tells the driver to map specified 64KB tiles within the surface to memory, within resources called Tile Pools. Tiles that are not mapped to a Tile Pool are null tiles.

Tiled Resource Translation Table (TRTT) is constructed as a 3 level tile Table. Each tile is 64KB in size which leaves behind 44-16=28 address bits. 28bits are partitioned as 9+9+10 which corresponds to TRVATT L3, L2 and L1 respectively. This is where TRVATT L3 has 512 entries, L2 has 512 entries and L1 has 1024 entries where each level is contained within a 4KB page hence L3 and L2 is composed of 64b entries and L1 is composed of 32b entries.



The contents of the TRVATT tables are as listed above where L3 and L2 points to the address of the next level which is a 4KB page and L1 contains the 32b VA address pointer needed to map the TR tile to virtual address space.

L1 Entry:

Bits	Field	Description
31:0	ADDR: Address	GFX virtual address of 64KB tile is referenced by this entry. This field is treated as GFX Virtual Address (GVA) when translated and maps to 47:16.

L2 Entry:

Bits	Field	Description
63:48	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
47:12	ADDR: Address	GFX virtual address or Guest Physical Address of 4KB base address pointing to TR-TT L1. *TR-TT table entries for L2 and L3 can be in GFX virtual address mode or Guest Physical address mode chosen by GFX software.*
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	Null	Null Tile where reads to this tile returns zero with a Null indicator and writes are dropped.
0	Invalid	Invalid Tile where reads to this tile returns zero and writes are dropped. Additional interrupt is generated to GFX software when an invalid tile is accessed.

L3 Entry:

Bits	Field	Description
63:48	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
47:12	ADDR: Address	GFX virtual address or Guest Physical Address of 4KB base address pointing to TR-TT L2. *TR-TT table entries for L2 and L3 can be in GFX virtual address mode or Guest Physical address mode chosen by GFX software.*
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	Null	Null Tile where reads to this tile returns zero with a Null indicator and writes are dropped.
0	Invalid	Invalid Tile where reads to this tile returns zero and writes are dropped. Additional interrupt is generated to GFX software when an invalid tile is accessed.

Programming Note	
Context:	Tiled ResourceTranslation Tables in Gfx Page Tables
GFX Driver has to disable the TR-TT bypass mode before using tiled resoruces translation tables. Details of the registers are given in "registers for TR-TT management."	

Programming Note	
Context:	Tiled ResourceTranslation Tables in Gfx Page Tables
GFX Driver is not allowed to put TR-TT entries into TR-VA space.	

Programming Note	
Context:	Tiled ResourceTranslation Tables in Gfx Page Tables
Usage model for TR translations are restricted to GFX Render Engine (& POSH pipeline).	

Programming Note	
Context:	Tiled ResourceTranslation Tables in Gfx Page Tables
TRTT is only for PPGTT64 (Advanced or Legacy PPGTT64). Enabling TRTT in Legacy PPGTT32 context or GGTT context is considered as invalid programming.	

Registers for TR-TT Management

Following register is a global mechanism to disable the bypass mode which is considered to be default for h/w. GFX driver has to set this bit to disable bypass mode before using TR-TTs.

Following registers shall be part of the h/w context.

Tiled Resources VA Translation Table L3 Pointer		
Register Space:	MMIO: 0/2/0	
DWord	**Bit**	**Description**
1	63:48	**Reserved** **Access:** RO Reserved.
	47:32	**Tiled Resource – VA translation Table L3 Pointer (Upper Address)** **Default Value:** 0000h **Access:** R/W Upper address bits for tiled resource VA to virtual address translation L3 table. For physical memory option, address bits [47:39] has to be programmed to "0" as it is defined the limit of physical memory allocation.
0	31:16	**Tiled Resource – VA translation Table L3 Pointer (Lower Address)** **Default Value:** 0000h **Access:** R/W Lower address bits for tiled resource VA to virtual address translation L3 table.
	15:0	**Reserved** **Access:** RO Reserved.

Tiled Resources Null Tile Detection Register		
Register Space:	MMIO: 0/2/0	
DWord	**Bit**	**Description**
	31:0	**Null Tile Detection Value** **Default Value:** 00000000h **Access:** R/W A 32bit value programmed to enable h/w to perform a match of TR-VA TT entries to detect Null Tiles. Hardware will flag each entry and space behind it as Null Tile for matched entries.

Tiled Resources Invalid Tile Detection Register		
Register Space:	MMIO: 0/2/0	
DWord	**Bit**	**Description**
	31:0	**Invalid Tile Detection Value** **Default Value:** 00000000h **Access:** R/W A 32bit value programmed to enable h/w to perform a match of TR-VA TT entries to detect Invalid Tiles. Hardware will flag each entry and space behind it as Invalid Tile for matched entries.

Tiled Resources Virtual Address Detection Registers (TRVADR)		
Register Space:	MMIO: 0/2/0	
DWord	**Bit**	**Description**
0	31:8	**Reserved** **Access:** RO Reserved.
	7:4	**TRVA Mask Value (TRVAMV)** **Default Value:** 0000b **Access:** R/W 4bit MASK value that is mapped to incoming address bits[47:44]. MASK bits are used to identify which address bits need to be considered for compare. If particular mask bit is “1”, mapping address bit needs to be compared to DATA value provided. If “0”, corresponding address bit is masked which

Tiled Resources Virtual Address Detection Registers (TRVADR)		
		makes it don't care for compare (*this field defaults to "0000" to disable detection*) *Note that h/w supports two possible values for MASK: "0000" which is disabled case and "1111" where 44 bit TR-VA space is carved out.*
	3:0	**TRVA Data Value (TRVADV)** **Default Value:** 0b **Access:** R/W 4bit DATA value that is mapped to incoming address bits[47:44]. Data bits are used to compare address values that are not filtered by the TRVAMV for match.

Tiled Resources Translation Table Control Register (TRTTE)		
Register Space:		MMIO: 0/2/0
DWord	**Bit**	**Description**
0	31:2	**Reserved** **Access:** RO Reserved.
	1	**TR-VA Translation Table Memory Location** **Default Value:** 0b **Access:** R/W This fields specifies whether the translation tables for TR-VA to VA are in virtual address space vs physical (GPA) address space. 0: Tables are in Physical (GPA) Space 1: Tables are in Virtual Address Space **Tiled Resource Translation Tables in GPA space is not supported in any GEN generations. this mode should never be set as GPA mode (always set to '1). HW will set TRTT tablesin Virtual address space mode only.**

Tiled Resources Translation Table Control Register (TRTTE)		
	0	**TR-TT Enable** **Default Value:** 0b **Access:** R/W TR translation tables are disabled as default. This field needs to be enabled via s/w to get TR translation active.

The following register (0x4DFC[0]) has enable and disable control of the bypass path across TR translations. By default, bypass is enabled, and bypass needs to be disabled (by setting 0x4DFC[0] = '1) for TR translations to function. Disabling the bypass should be done before render power gating is enabled.

Detection and Treatment of Null and Invalid Tiles

Two types of definition that need to be extracted from TR-VA walk in addition to reaching the GFX virtual address.

1. **Null Tiles**: Null tiles provide the applications the of capability to preventing OS mapping the entire surface. When a memory access hits a Null tile, the access is terminated and zero's are returned to the originator of the memory access for loads along with a null indicator and for stores the access is dropped at the page walker level.
2. **Invalid Tiles**: This is the case where GFX software did not update the value of the mapping properly for hardware to separate resident vs null tiles. The Invalid Tile treatment is exactly same however additionally a unique interrupt is generated in h/w

Both detections are done by GPU:

- For L2/L3 entries, Null and Invalid tile information is already embedded in the TR-TT entries
- For L1 entries, the contents (32bits) are compared in hardware to pre-programmed values by GFX software (*values are provided in GFX MMIO space*). For the match values, two separate 32b registers are defined, one for Null Tile detection and one for Invalid Tile detection.

Hardware walking matching the value or detecting L2/L3 would terminate the walk (i.e. rest of the tables are not valid) and define the access as either Null or Invalid.

Programming Note	
Context:	Detection and treatment of null and invallide tiles.
The software is not allowed to program both Null and Invalid values to be the same.	

Programming Note	
Context:	TileX Surfaces and Null Tiles
NULL or Invalida Tiles are not supported on TileX surfaces.	

GPU implements a counter mechanism to roll-up the Null tile accesses detected. The counter value is exposed to GFX software via GFX MMIO.

When the TR translation tables are in Gfx virtual address domain, the pages faults encountered while walking the IA32e pages are not reported back to the TR walkers or TLBs. These faults are handled as fault & halt, making these faults transparent to the TR walkers. However, when such a fault is not fixed (unsuccessful fault response) or when a non-recoverable fault encountered, main page walker HW convertes the cycle to an invalid cycle. Thus, in this case, TR walker or TR TLBs will get incorrect read return data without any notification of the non-recoverable fault condition. Thus TR walker/TLBs will continue with the TR-walk with incorrect data. This can lead to spurious cycles being generated. However, a Gfx reset/FLR is expected as a result of the non-recoverable fault.

TR-TT Modes

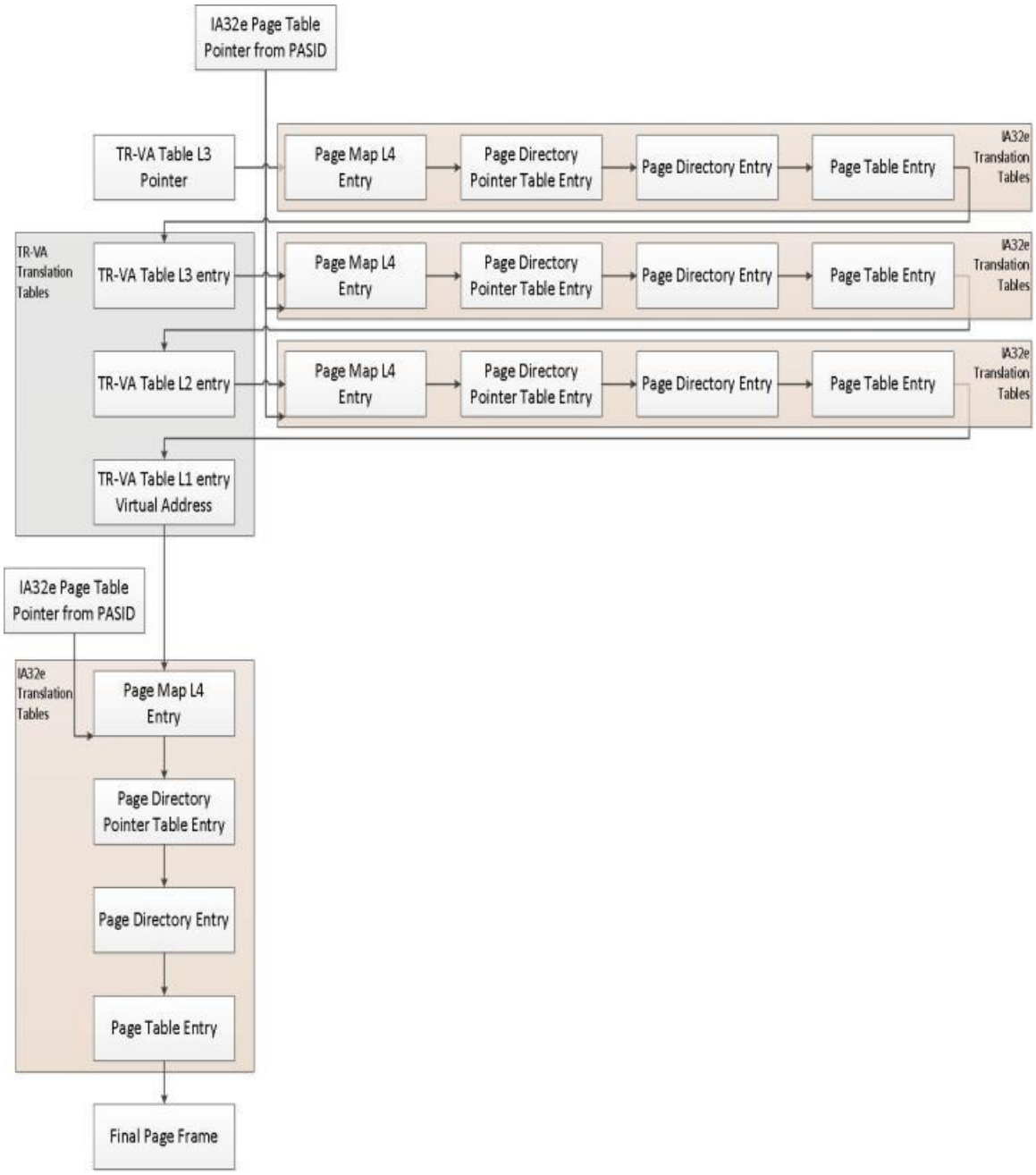
The L3 table pointer along with TRTTL3e/TRTTL2e is projected to support two modes of address space. Original intent was to have the contents to be in Virtual Address space (OS managed) and have them to be translated to GPA to HPA before getting accessed. Such mechanism will incur high latency penalties due to nested page translations. GPU shall have an additional mode where tiled-resources translation tables are in physical address space (GPA) and eliminate the need to have nested translations to reduce the potentially high miss latencies.

TR-TT walker shall have both modes supported. The Mode bit will be part of the same Register that provides TR-VA TT L3 pointer.

Virtual Addressed TR Translation Tables

Having sparse tiled resource translation tables in GFX virtual space requires the h/w TR-TT walker to walk thru the 1st level tile tables for table accesses to reach to Physical address at the L1 TR translation tables.

The following diagrams provide the view of the walk TR-VA translation tables are in physical memory and no 2nd Level (VTd) translations enabled.



Once 2nd level translations are enabled each level of 1st level walk needs to be further walked through VTd page tables.

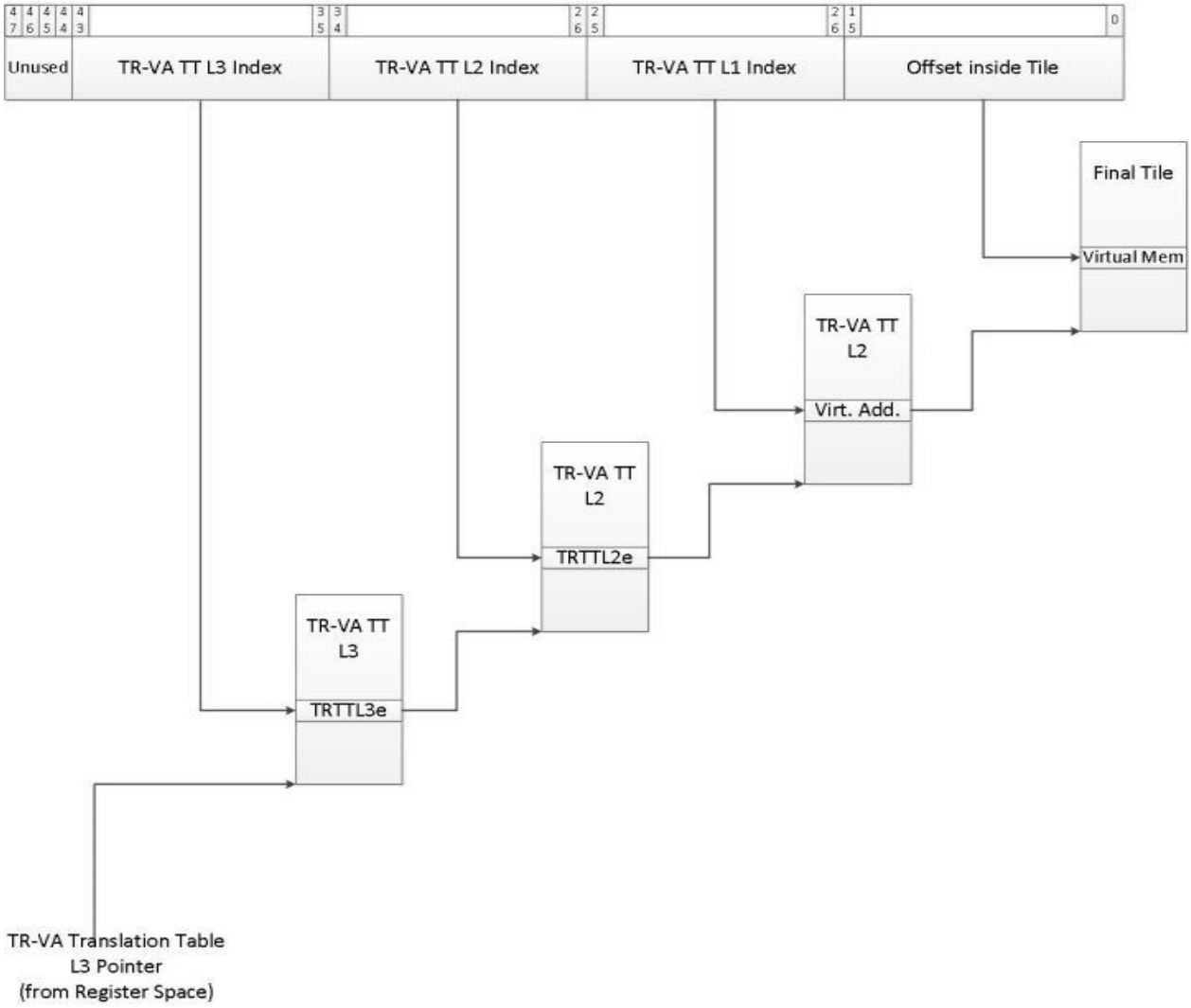
The level of nested walks does not change the structure of the TR-VA walker; it just defines the recursive nature of the translations.

TR-TT Page Walk

Sparse Tiled Resources translation tables are separated into 3-levels. The pointer to L3 table is going to be set up in GFX MMIO space as part of the context, this pointer be would be available to page walker ahead of any TR-VA memory accesses.

TR-TT L3 walk will be consistent of calculating the 64b of interest based on the L3 table pointer and using the 9 bit index (address bits[43:35]). L2 will use TR-TT L3 entry as the table pointer and use the next set of 9 address bits ([34:26]) to locate the L2 entry which is a pointer to L1 table. Final L1 table is located with L2 entry and indexed by remaining 10 address bits (25:16) to index where 32b virtual address is extracted.

Post TR-TT walk 32b entry from L1 is mapped to final virtual address 47:16 and remaining 15:0 is passed from the original TR-VA access as is given all tiles in TR-VA space are 64KB in size.



Page Table Modes

GFX Aperture and Display accesses are mapped thru Global GTT to keep the walk simple (i.e. 1-level) and latency sensitive. GPU accesses to memory can be mapped via Global GTT and/or ppGTT with various addressing modes.

Supported walk modes are listed as following:

1. **Global GTT with 32b virtual addressing**: Global GTT usage is similar to previous generations with extended capability of increasing virtual address (VA) up to 4GB (from 2GB) and use a standard 64b PTE format. The breakdown of the PTE for global GTT is given in later sections and allows 1-level page walk where the 20b index is used to select the 64b PTE from memory.
2. **Legacy 32b VA with ppGTT**: This is a mode where ppGTT page tables are considered private and managed via GFX sotfware (driver) where context is tagged as Legacy 32b VA. Each page walk is managed via 9b of the virtual address and 20b index to address 4GB memory space is broken into 3 parts. In order to optimize the walks and make it look like previous generations, GFX sotfware provides 4 pointers to page tables (called 4 PDP entries) all guest physical address. GPU uses the four pointers and fetches the 4x4KB into h/w (for render and media) before the context execution starts. The optimization limits the dynamic (on demand) page walks to 1-level only.
3. **Legacy 48b VA with ppGTT**: GFX address expansion beyond 4GB is added to address 48b virtual address space. 48b VA requires 36b indexing (4x9b) translating into 4-levels of page walk. To reduce the overhead of 4 level walk, GPU will cache the entire content of PML4 (4kB) to limit the on-demand walks to 3 levels. The caching happens as part of the initial demand where no further replacements required.
4. **Advanced 48b VA with IA32e support via IOMMU**: 48b addressing in advanced mode is managed via IOMMU settings where the base of the page table shall be found after the root / context tables using bus/device/function values. PASID# is used as an index in PASID table to find page table pointer to start the 4-level page walk. Rest of the mechanism is similar to Legacy 48b VA mode, GPU has the capability to cache entire content of PML4 and try to limit the dynamic page walks to 3-level.

Per Process GTT

GTT mechanism has multiple hooks and mechanisms for s/w to prepare the page walks on hardware. The listed mechanisms here are selectable per-context and descriptors are delivered to hardware as part of context descriptor.

The entry contents are also modified to match the same format as IA32e page tables allowing future expansion for sharable page tables as well as higher order virtual addressing.

Page Tables Entry (PTE) Formats

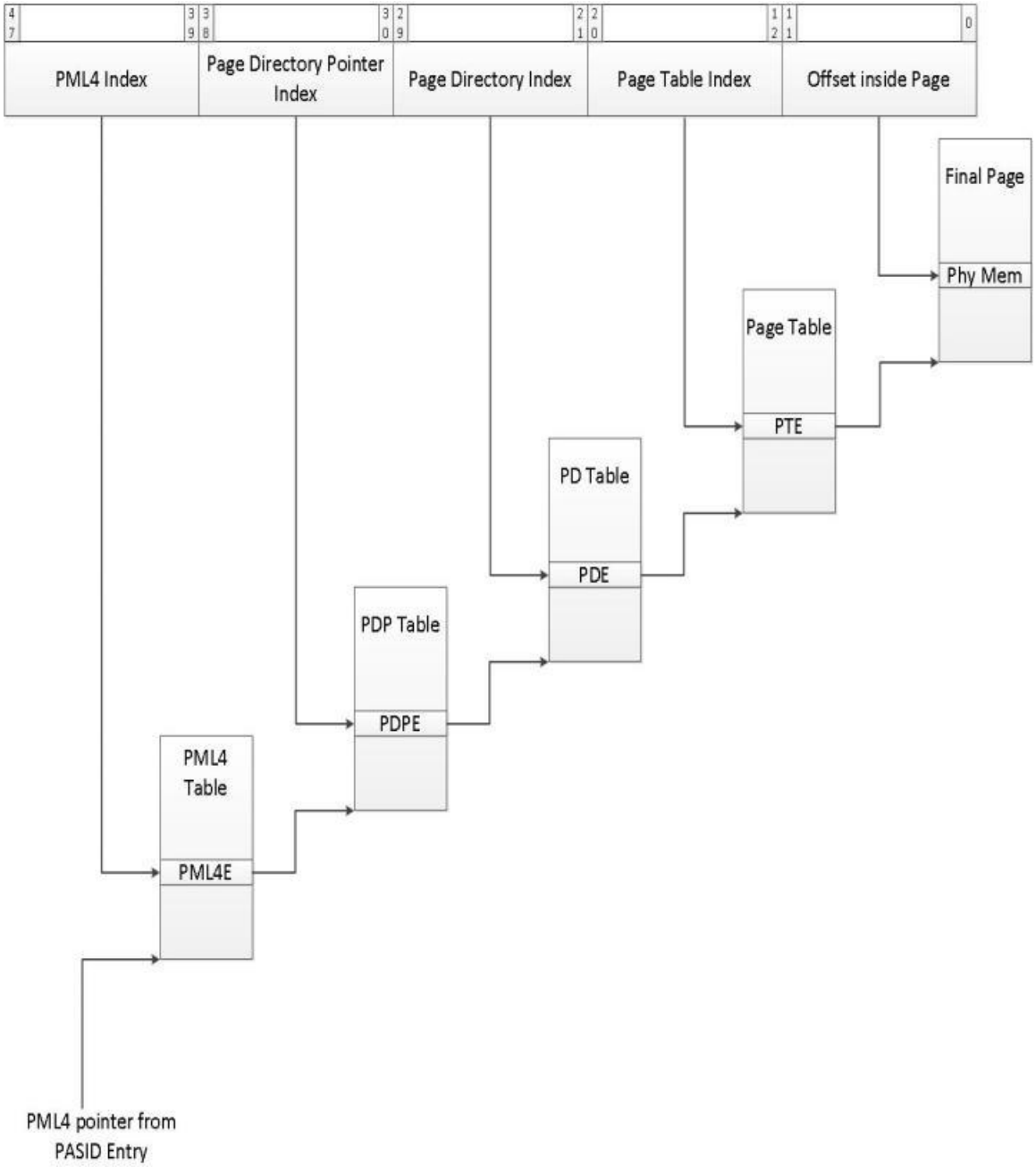
Page Table Entry (PTE) formats follow the IA32e layout shown below. Note that the Hardware Address Width (HAW) is determined by Uncore: typically 39 for client products and 46 for server products.

63	62:61	60:59	58:57	56:55	54:53	52:51	HAW	HAW-1	31:30	29:28	27:26	25:24	23:22	21:20	19:18	17:16	15:14	13:12	11:10	9	8	7	6	5	4	3	2	1	0	
XD	Ignored						Rsvd.		Address of page-directory-pointer-table											Ign	EA	Ign	Rsvd	Ign	A	PCD	PWT	U/S	R/W	P	PML4E
XD	Ignored						Rsvd.		Address of the 1GB page									Reserved		PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P	PDPE (1GB Page)
XD	Ignored						Rsvd.		Address of page-directory-table											Ign	EA	Ign	0	Ign	A	PCD	PWT	U/S	R/W	P	PDPE Page Directory
XD	Ignored						Rsvd.		Address of the 2MB page									Reserved		PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P	PDE (2MB Page)
XD	Ignored						Rsvd.		Address of page-table											IPS	EA	Ign	0	Ign	A	PCD	PWT	U/S	R/W	P	PDE Page Table
XD	Ignored						Rsvd.		Address of the 64KB MB page									Rsvd.		Ign	EA	Ign	G	PAT	D	A	PCD	PWT	U/S	R/W	P	PTE (64KB Page)
XD	Ignored						Rsvd.		Address of 4KB Page											Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P	PTE (4KB Page)

Each table entry is further broken down along with the required functions. GFX has a 4-level page table which is pointed out by context descriptor starting with the 4th level of PML4. The next levels have slightly different formats depending on the size of the page supported. 1GB and 2MB page formats are required for support.

Page walk in advanced mode with 48b VA requires 4 levels. The walk will start with a PML4 table pointer extracted from PASID entry and uses the 48b VA as index to consecutive levels of page tables.

The following diagram shows the page walk that is needed for a 4KB page:



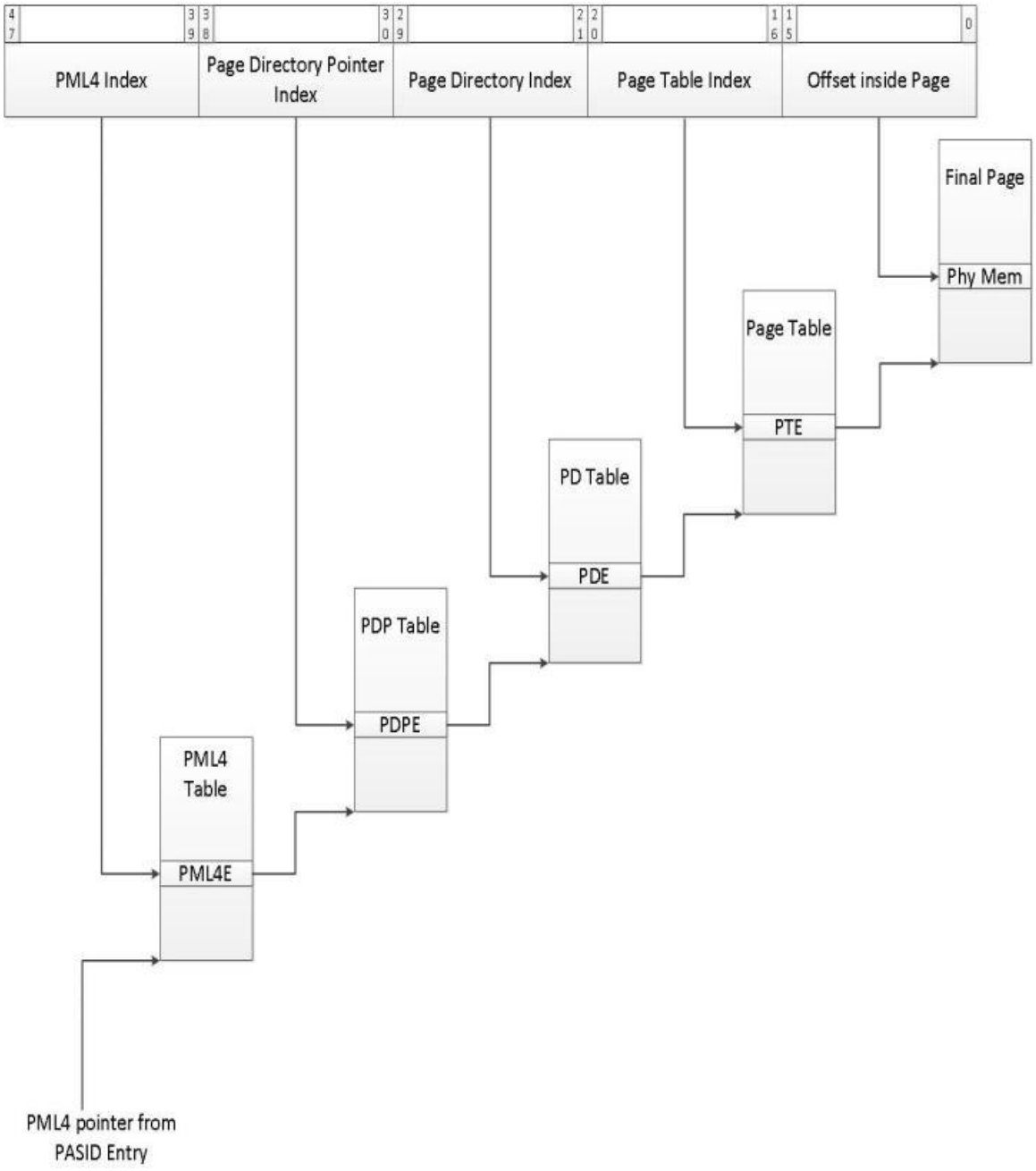
A 64 bit (48b canonical) address requires 4 levels of page table format where the context carries a pointer to highest level page table (PML4 pointer) via PASID. The rest of the walk is normal page walk thru various levels.

To repurpose the caches the following mechanism is used:

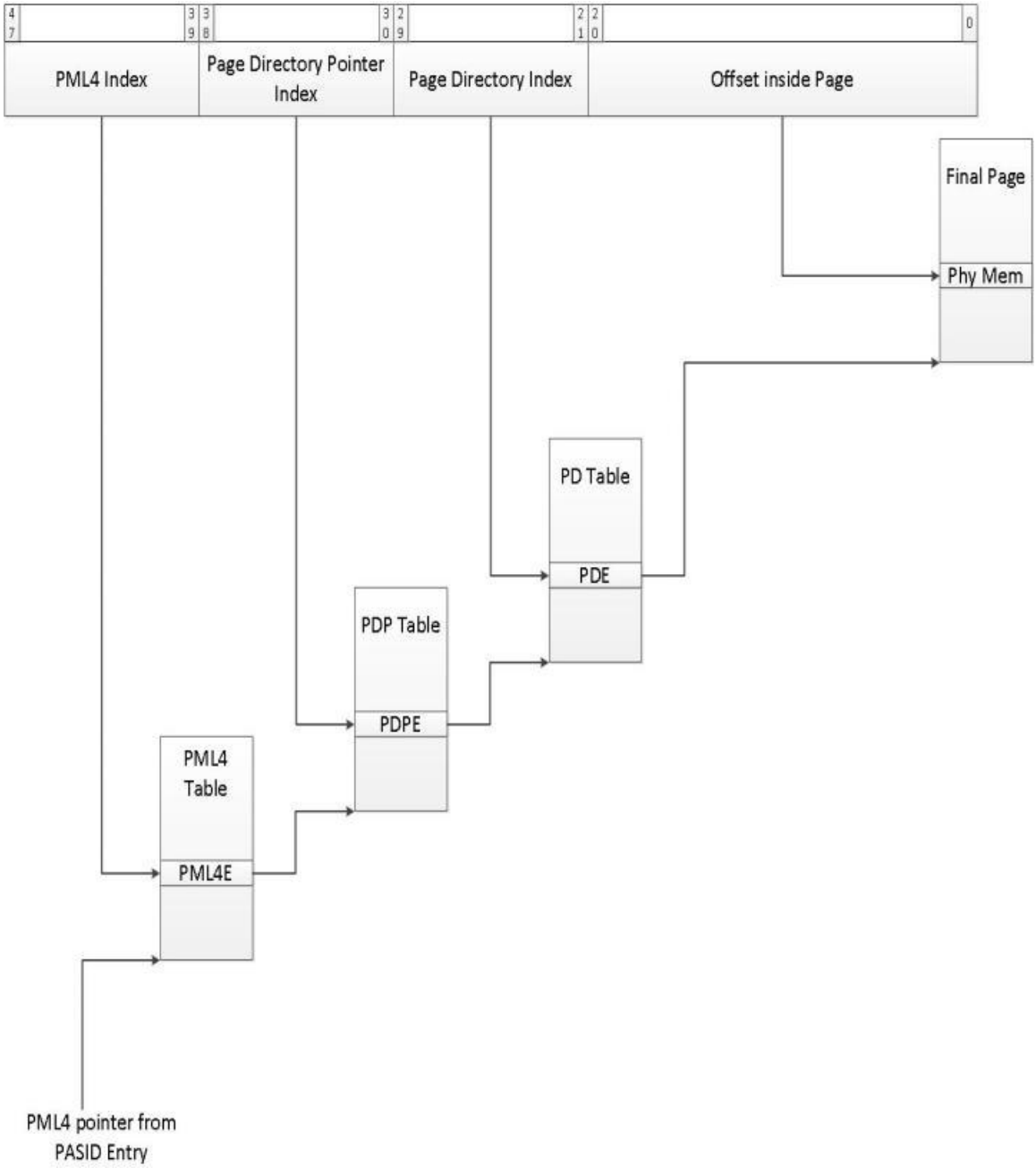
- 3D: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- Media: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- VEBOX, Blitter: each with 4KB acting as PML4, PDP, PD cache.

The design sections the 512 entries within 4KB into separate areas for PML4, PDP, and PD.

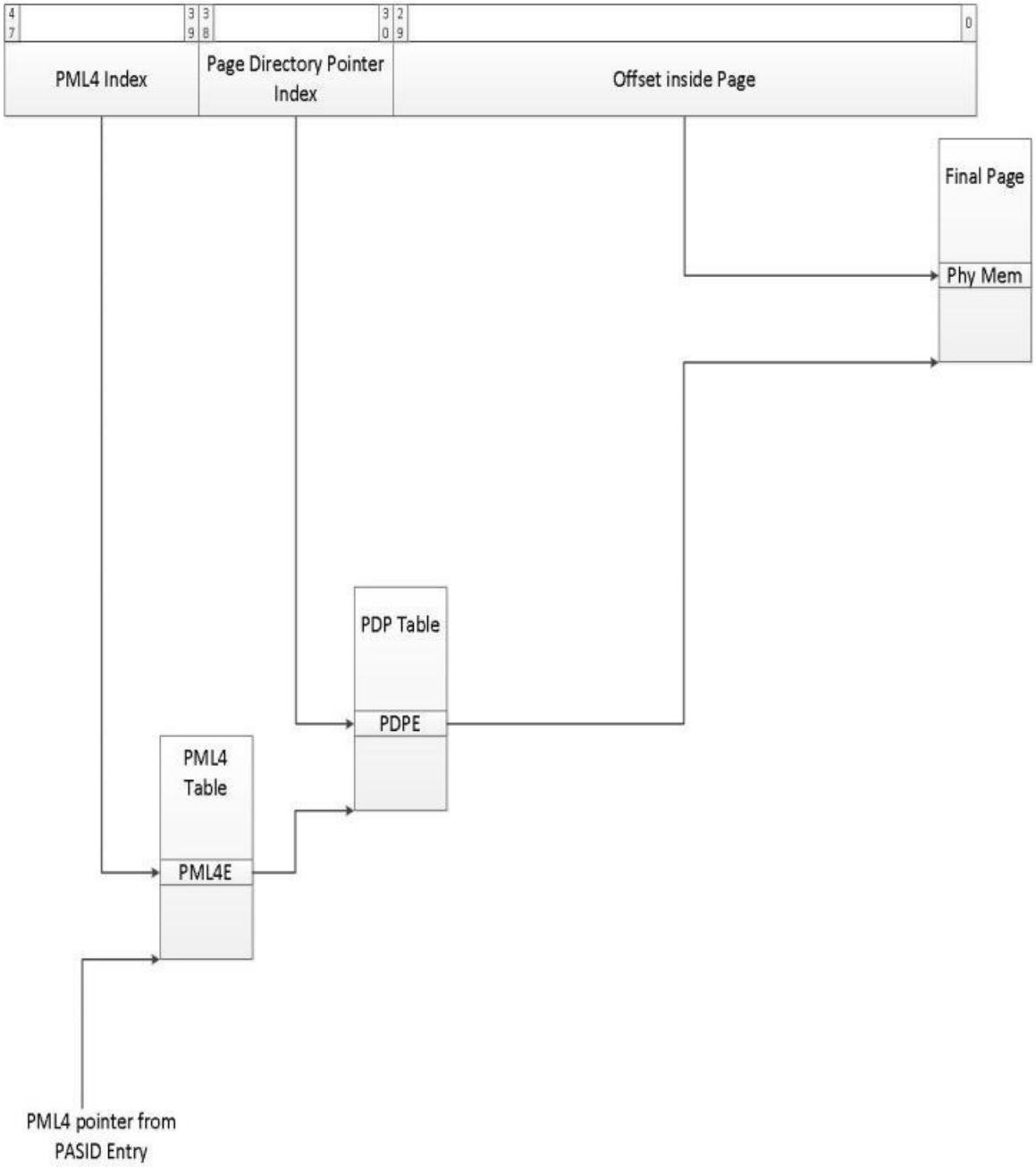
The 64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In a page table every 16th entry (PTE#0, PTE#16, PTE#32, ... PTE#496) should be used to index. This is calculated using address[20:16] & "0000". Note that hardware should not make any assumptions for any other PTEs.



With the 2MB Page walk, the last level of the page walk is skipped where the PD entry points to the final page.



For the support for 1GB page size, the following mechanism is needed.



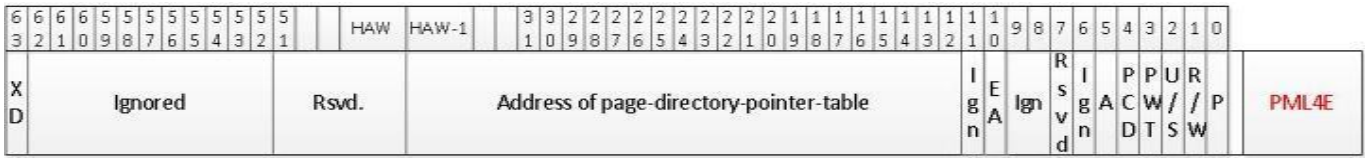
Pointer to PML4 Table

Page table pointer is the starting address where the PML4 table starts. The contents of pointer will be provided by PASID table entry in case of advanced context, else it will be provided by software as part of the legacy context with 48b addressing.

Details of PASID entry is given in later sections.

PML4E: Pointer to PDP Table

PML4 is used to locate the page directory pointer tables distributed in physical memory. PML4 will be used for advanced GPGPU context scheduled via PASID table as well as legacy context with 48b VA.



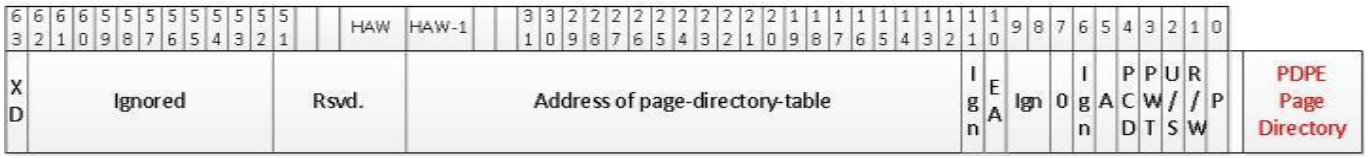
Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*

Bits	Field	Description
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PML4 Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDPE: Pointer to PD Table

PDP entry is used to locate the base of the PD table:



Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.

Bits	Field	Description
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PDP Entry is present. It must be "1" to point to a page directory pointer table

PDP entry for 1 GB Page

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):30	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
29:13	Reserved	*Reserved (must return 0's)*
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
11	Ignored/Reserved	Ignored/not used by hardware

Bits	Field	Description
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	Page Size	Must be 1 to indicate 1GB page.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 1GB Page.

* HAW = 39 for client, and 46 for server.

PD: Pointer to Page Table

Page Directory entry has few different usage models:

1. It can identify the base of the page table.
2. It can define 2MB page table entries.

Pointer to page table is given below:

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	IPS	An MMIO level control has been introduced to manage 64KB page enabling.
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. *GPU does not support any memory type but WB when accessing paging structures.*
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*

Bits	Field	Description
0	P: Present	PD Entry is present. It must be “1” to point to a page directory pointer table

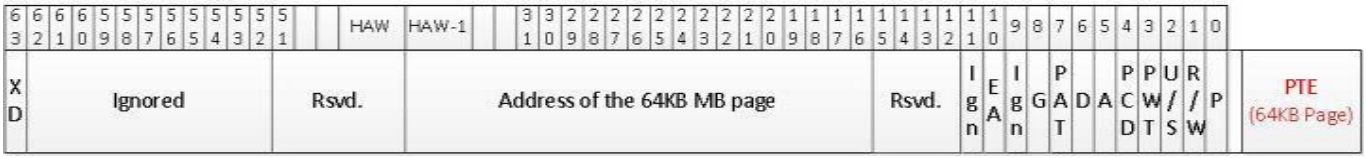
PDE for 2MB Page is given below:

Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0’s)*
(HAW-1):21	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
20:13	Reserved	*Reserved (must return 0’s)*
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	Page Size	Must be 1 to indicate 2MB page.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.

Bits	Field	Description
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 2MB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 64KB Page



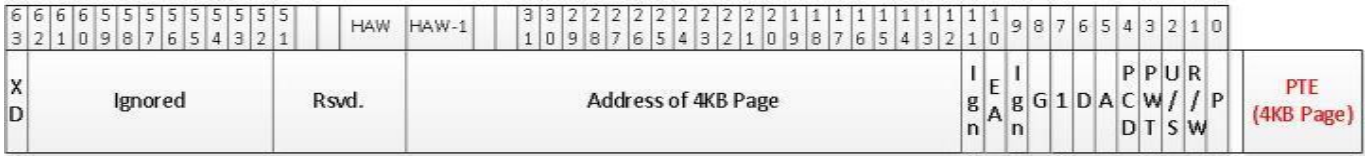
Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
15:12	Reserved	*Reserved (must return 0's)*
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*

Bits	Field	Description
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

64KB pages need to be enabled via MMIO along with the PDE IPS bit per directory entry.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 4KB Page



Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*

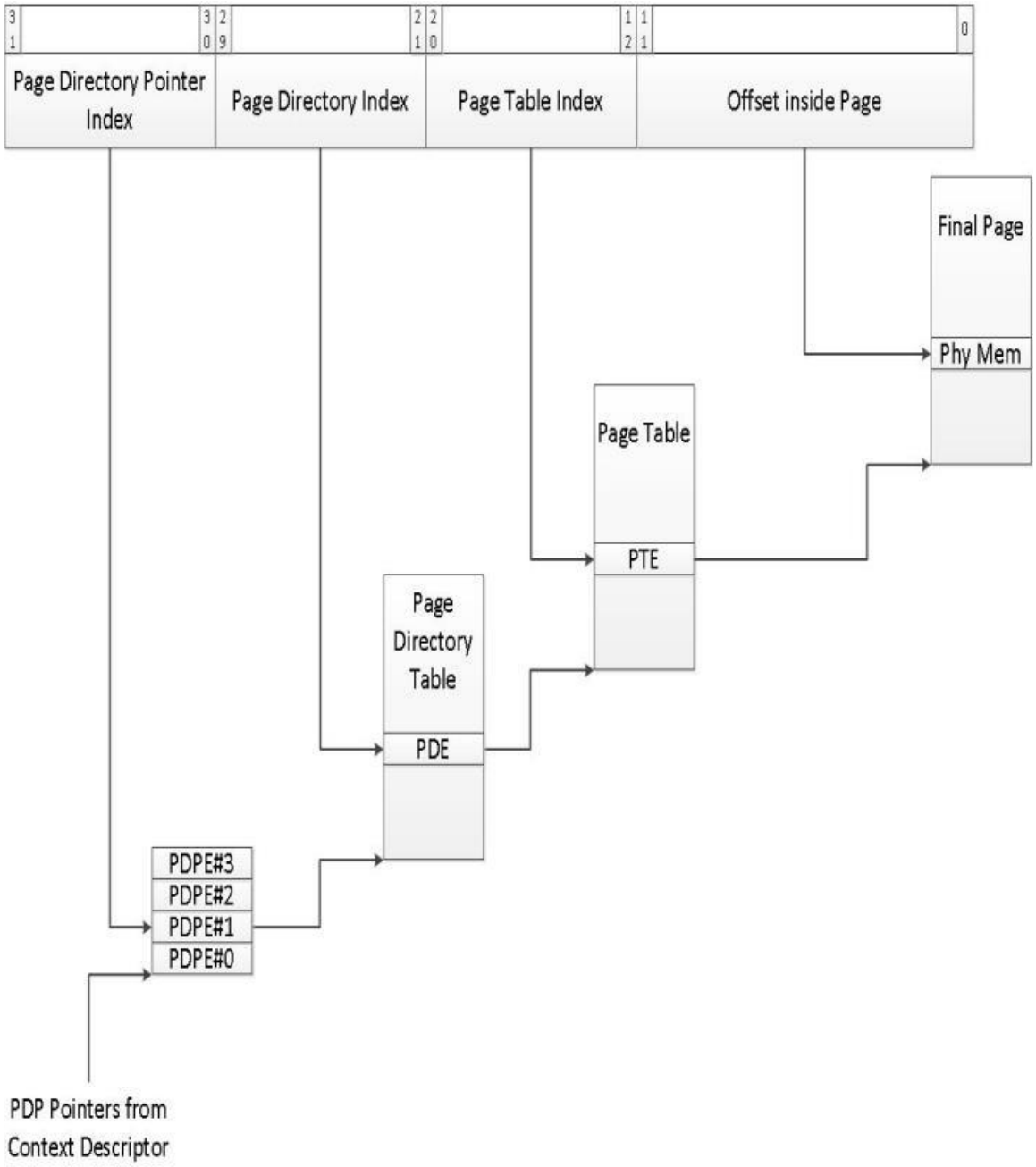
Bits	Field	Description
(HAW-1):12	ADDR: Address	Physical address of 4KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 4KB Page.

* HAW = 39 for client, and 46 for server.

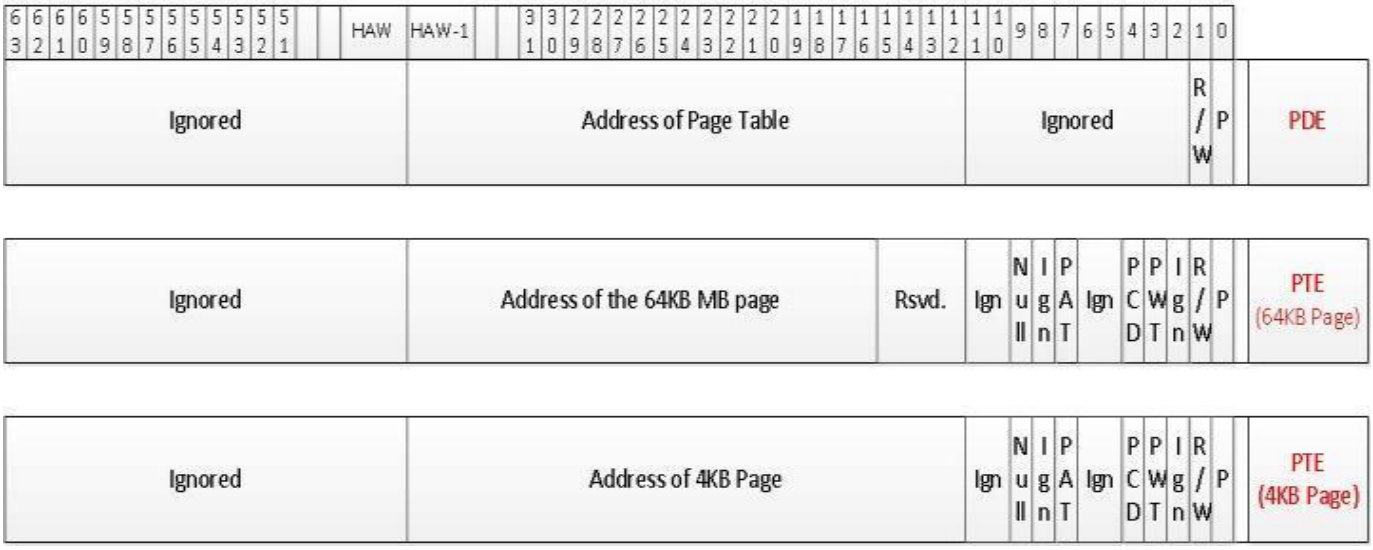
PPGTT for 32b Virtual Address

For page walk in legacy mode with 32b VA, we need two levels. The walk starts with a PDP pointer provided by the context descriptor, and uses the 32b VA as an index to consecutive levels of page tables. Hardware implements 16KB intermediate caches to limit the page walk needed to a single level, to have the same sensitivity to latency as previous generations.

The following diagram shows the page walk needed for a 4KB page.



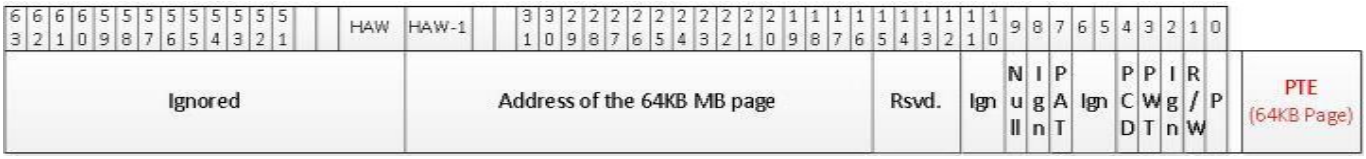
Page Table Entry formats for 32b VA use the following formats:



PDE for the page table

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests targeted to the memory range pointed by this PDE. *In Legacy mode with 32b VA, R/W bits from PDE are not used.*
0	P: Present	PD Entry is present. It must be "1" to point to a page directory pointer table

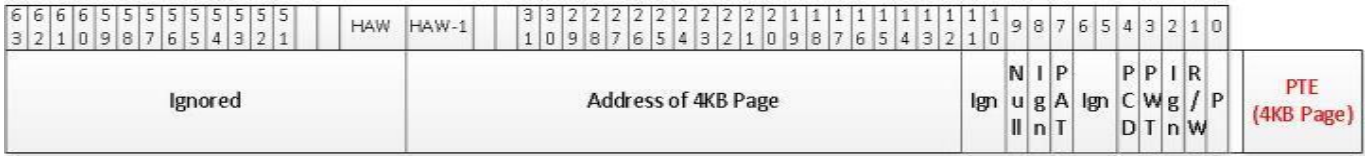
PTE for 64KB page



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.

Bits	Field	Description
15:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

PTE for 4KB Page



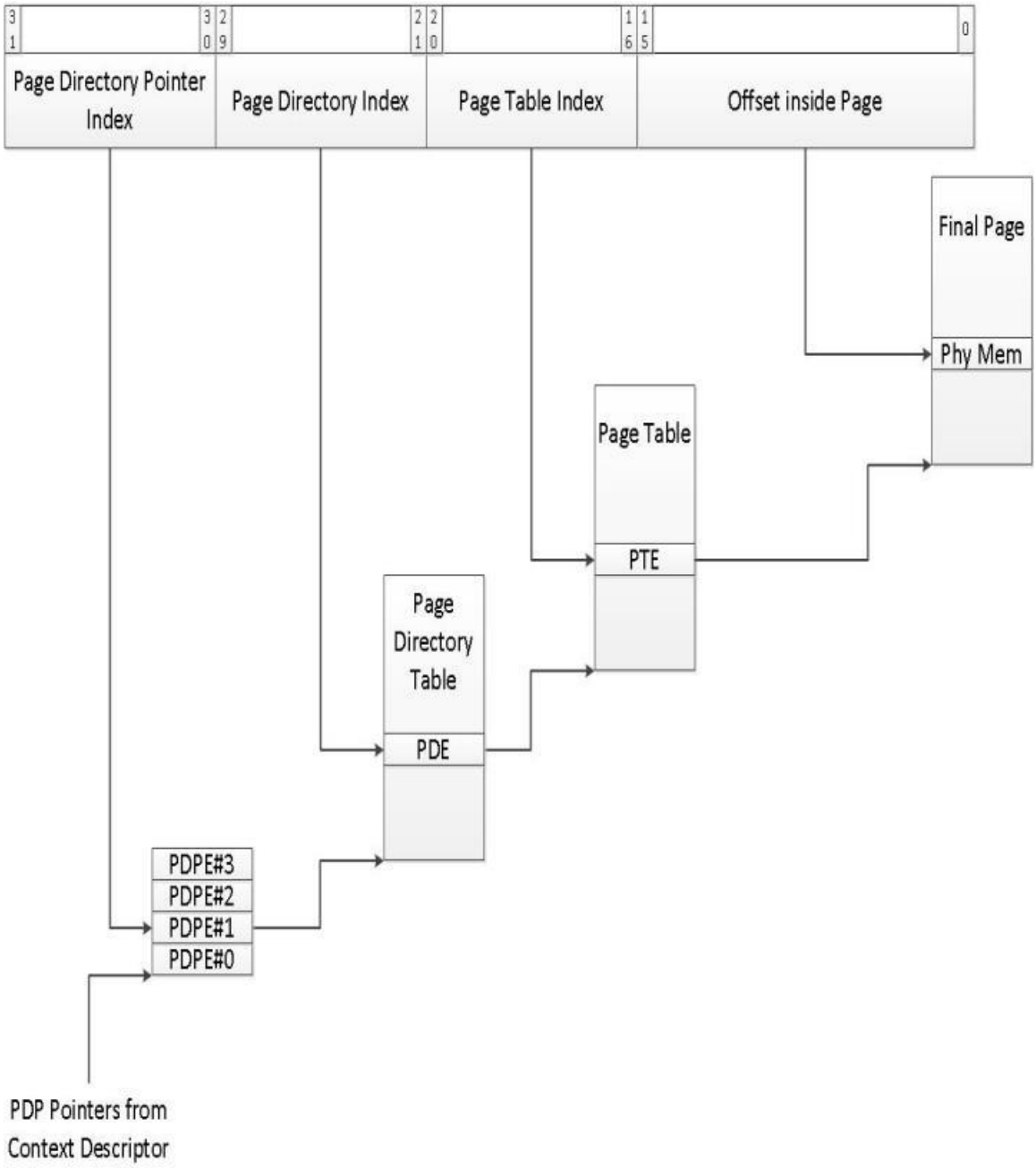
Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*

Bits	Field	Description
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

* HAW = 39 for client, and 46 for server.

Walk with 64KB Page

64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In page table every 16th entry (PTE#0, PTE#16, PTE#32….PTE#496) should be used to index. This is calculated using address[21:16] & "0000". Note that hardware should not make any assumptions for any other PTEs.



Walk with 2MB Page

PPGTT32 does not support 2MB pages.

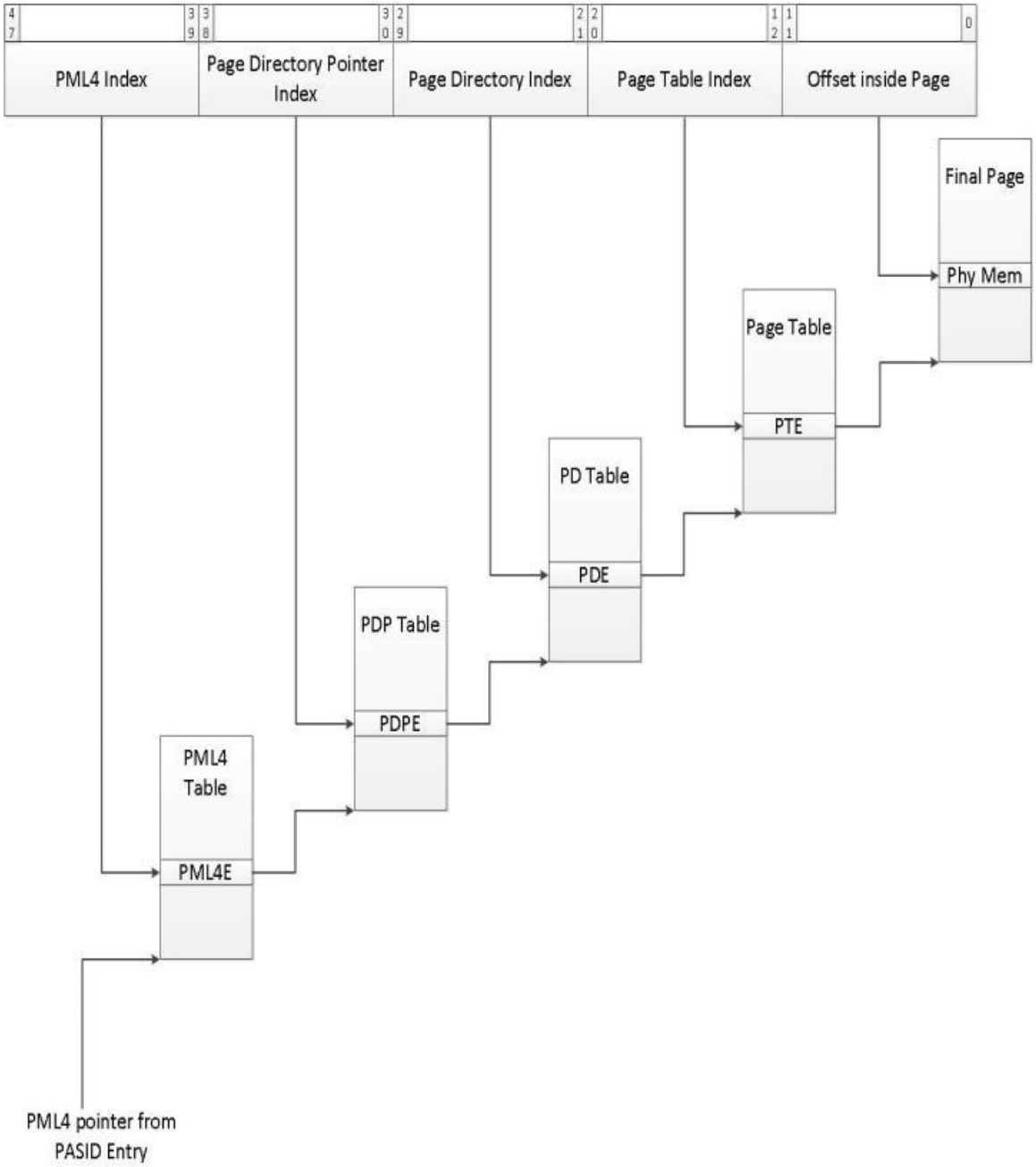
Walk with 1GB Page

PPGTT32 does not support 1GB pages.

PPGTT for Standard Context (64b VA)

For page walk in advanced mode with 48b VA, we need four levels. The walk starts with a PML4 table pointer given by GFX software and uses the 48b VA as index to consecutive levels of page tables.

The following diagram shows the page walk that is needed for a 4KB page:



A 64-bit (48b canonical) address requires 4-levels of page table format where the context carries a pointer to the highest level page table (PML4 pointer) via PASID. The rest of the walk is a normal page walk thru the various levels.

To repurpose the caches the following mechanism is used:

- 3D: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- Media: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- VEBOX, Blitter: each with 4KB acting as PML4, PDP, PD cache.

The design sections the 512 entries within 4KB into separate areas for PML4, PDP, and PD.

Page Table Entry (PTE) formats follow a similar layout to IA32e as given below.

63–52	HAW	HAW-1	51–12	11–9	8	7	6	5	4	3	2	1	0	
Ignored			Address of page-directory-pointer-table	Ignored								R/W	P	PML4E

63–52	HAW	HAW-1	51–30	29–13	12	11–9	8	7	6	5	4	3	2	1	0	
Ignored			Address of the 1GB page	Reserved	PAT	Ign	Null	Ign	1	Ign	PCD	PWT	Ign	R/W	P	PDPE (1GB Page)
Ignored			Address of page-directory-table			Ignored								R/W	P	PDPE Page Directory

63–52	HAW	HAW-1	51–21	20–13	12	11–9	8	7	6	5	4	3	2	1	0	
Ignored			Address of the 2MB page	Reserved	PAT	Ign	Null	Ign	1	Ign	PCD	PWT	Ign	R/W	P	PDE (2MB Page)
Ignored			Address of page-table			Ignored								R/W	P	PDE Page Table

63–52	HAW	HAW-1	51–16	15–13	12	11–9	8	7	6	5	4	3	2	1	0	
Ignored			Address of the 64KB MB page	Rsvd.	Ign	Null	Ign	PAT	Ign	PCD	PWT	Ign	R/W	P		PTE (64KB Page)

63–52	HAW	HAW-1	51–12	11–9	8	7	6	5	4	3	2	1	0	
Ignored			Address of 4KB Page	Ign	Null	Ign	PAT	Ign	PCD	PWT	Ign	R/W	P	PTE (4KB Page)

Each table entry is further broken down along with the required functions. GFX has a 4-level page table which is pointed out by context descriptor starting with the PML4. The next levels have slightly different formats depending on the size of the page supported. 1GB and 2MB page formats are required for support.

In 48b legacy mode, the pointer to the PML4 table is provided via the context descriptor provided by GFX software. The PML4 entry format is given below and points to the base of the PDP table.

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.* *In 64b Legacy, R/W in PML4 entry can not be used for RO pages.*
0	P: Present	PML4 Entry is present. It must be "1" to point to a page directory pointer table

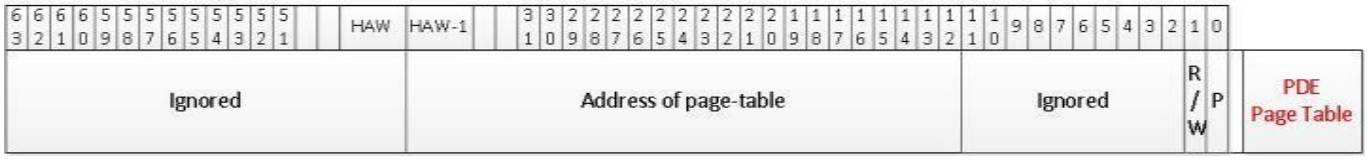
PDP entry is used to locate the page directory. Similar to IA32e page tables, legacy 48b VA supports 1GB pages, the PDPE has a mechanism to identify a way to say whether this PDPE represents a pointer to page directory or to a contiguous 1GB physical memory. PDP entry format is given below and points to the base of PD table.

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.* *In 64b Legacy, R/W in PDP entry can not be used for RO pages*
0	P: Present	PDP Entry is present. It must be "1" to point to a page directory pointer table

PDP entry for 1GB Page

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):30	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
29:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 1GB Page.

Page Directory entry point to the base of the page table and format is given below.



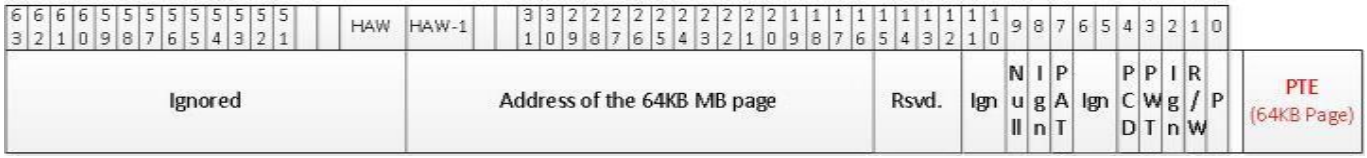
Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page- table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.* *In 64b Legacy, R/W in PD entry can not be used for RO pages*
0	P: Present	PDP Entry is present. It must be "1" to point to a page directory pointer table

Page Directory entry for 2MB page:

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):21	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
20:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.

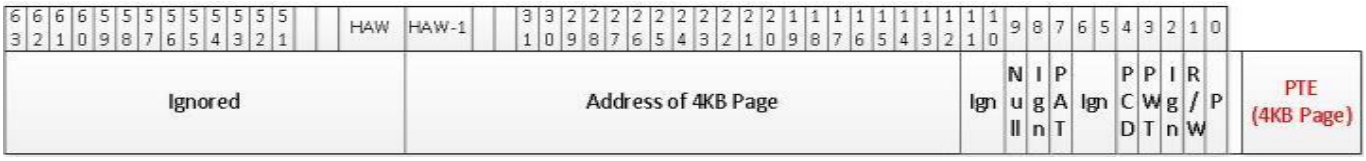
Bits	Field	Description
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 1GB Page.

Page Table entry for 64KB page:



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
15:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

Page Table Entry for 4KB page:

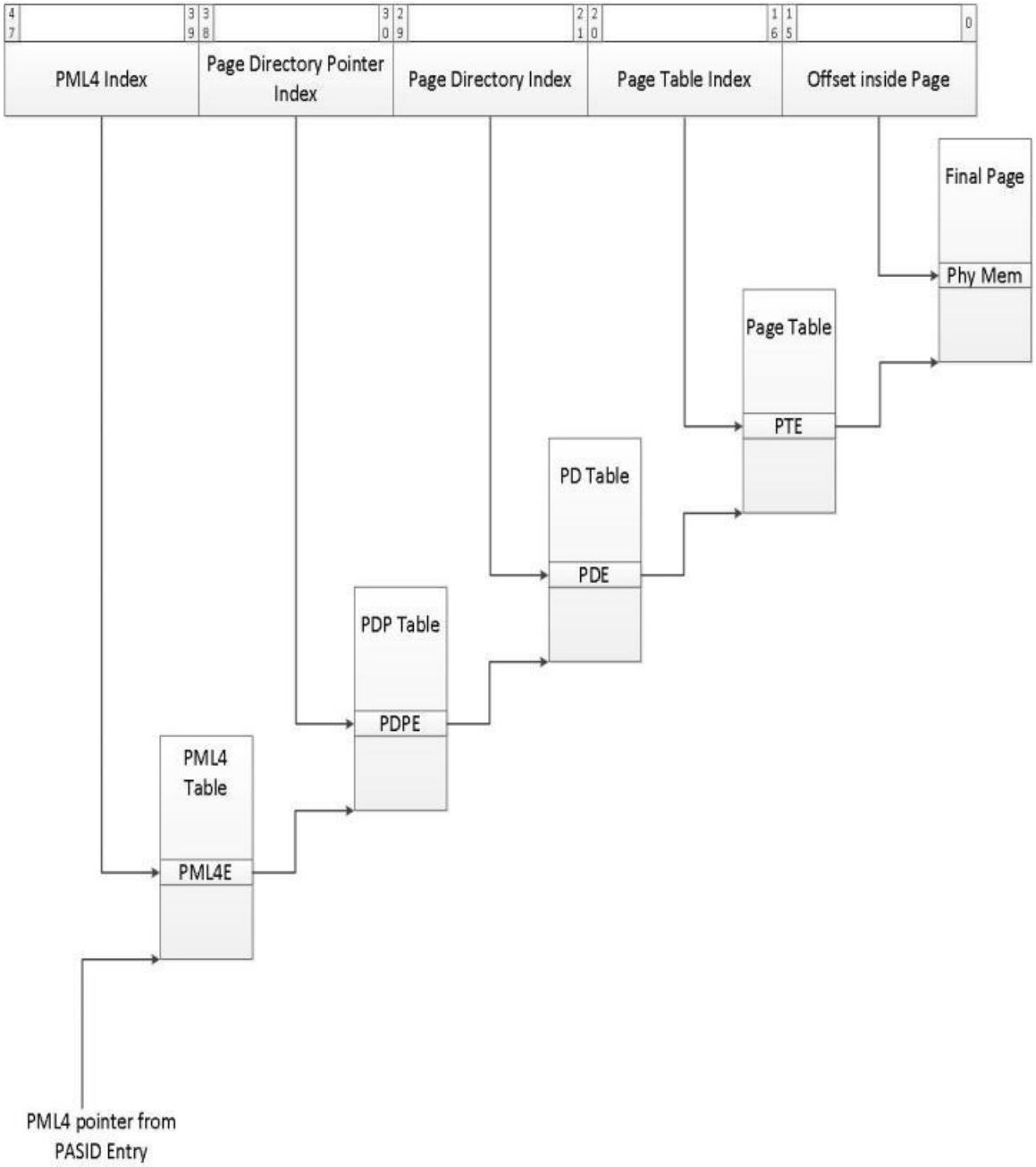


Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

* HAW = 39 for client, and 46 for server.

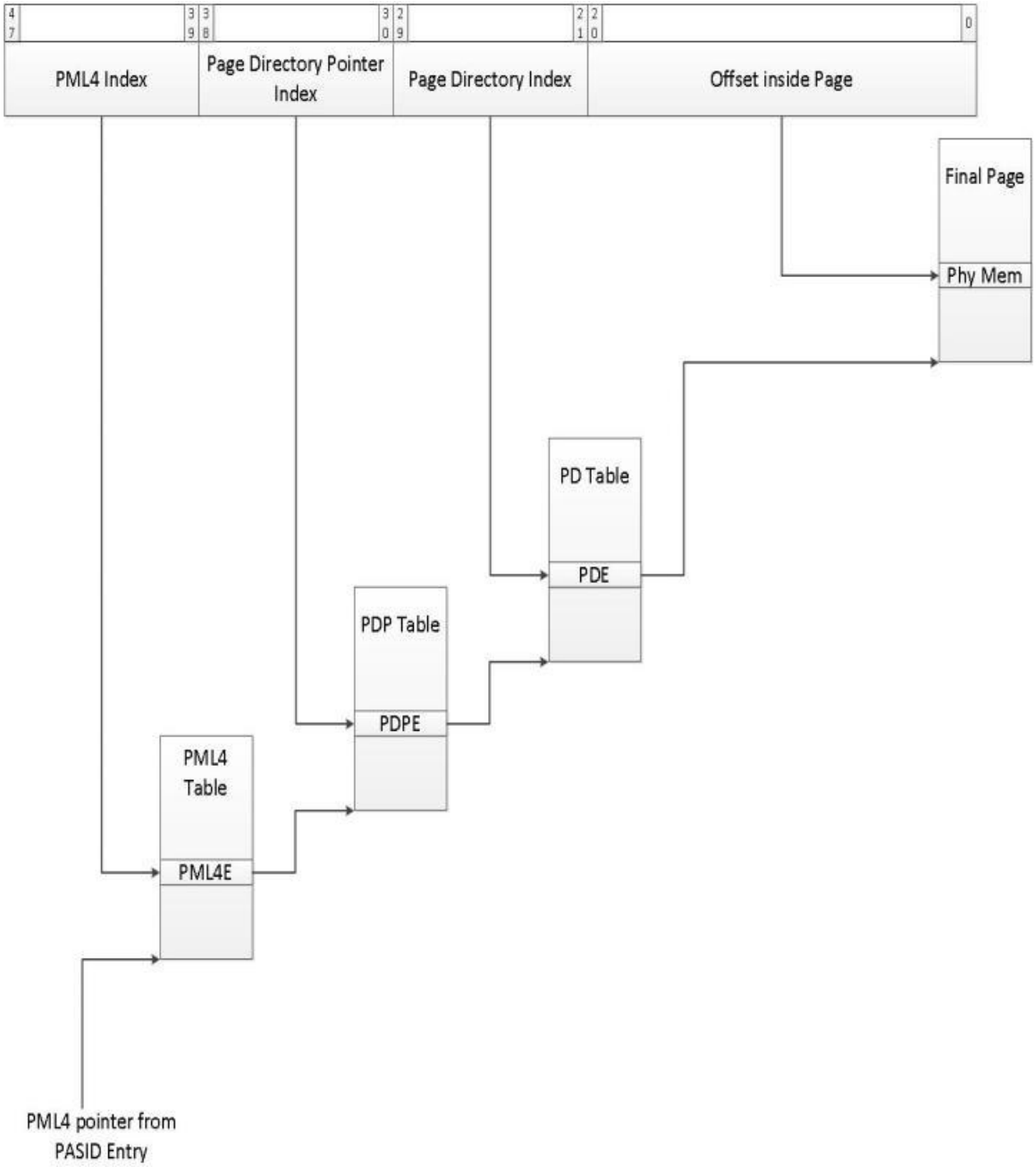
Walk with 64KB Page

64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In page table every 16th entry (PTE#0, PTE#16, PTE#32….PTE#496) should be used to index. This is calculated using address [20:16]& "0000". Note that hardware should not make any assumptions for any other PTEs.



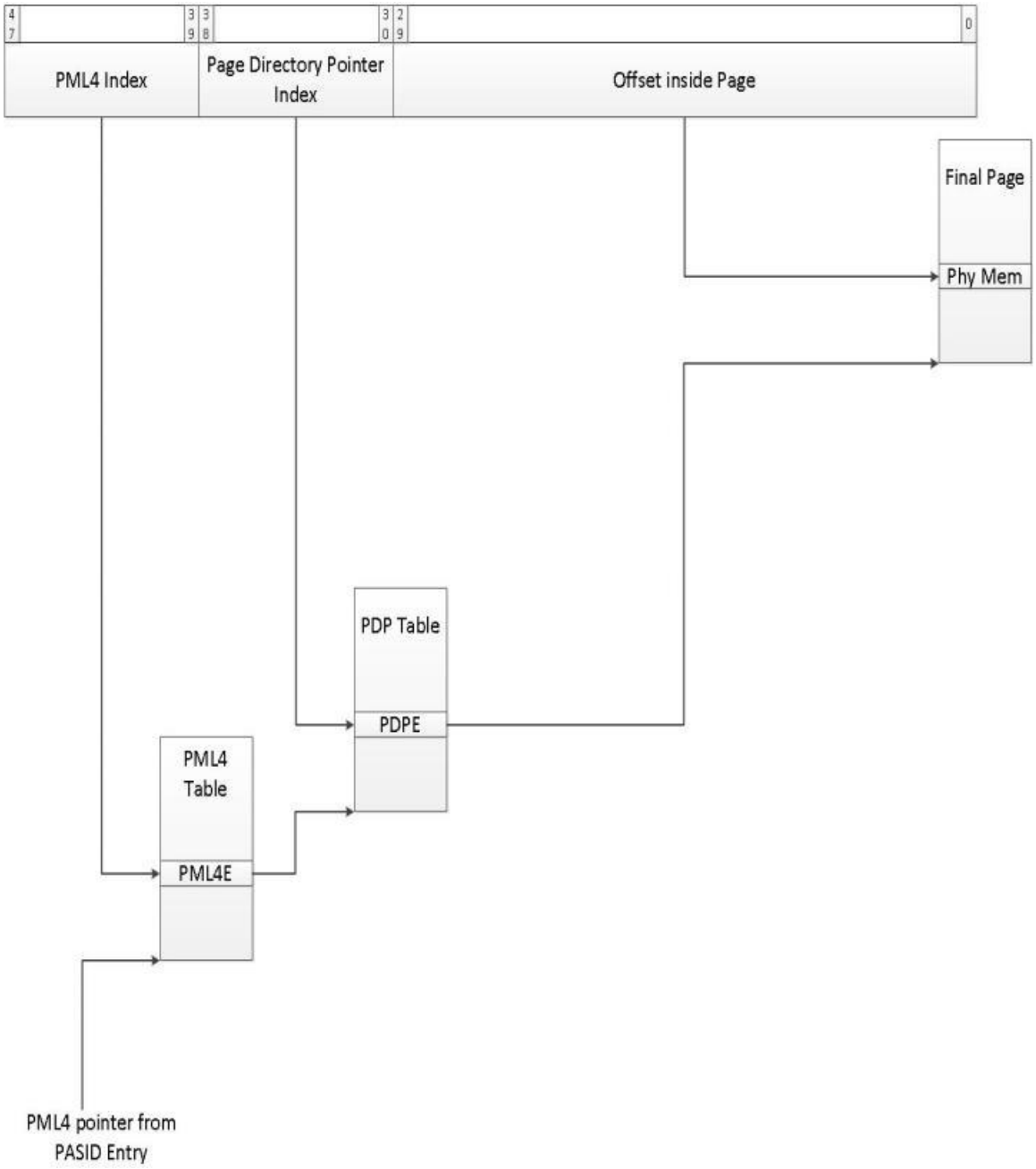
Walk with 2MB Page

With the 2MB Page walk, last level of the page walk is skipped where the PD entry points to the final page.


47
39
38
30
29
21
20
0
PML4 Index
Page Directory Pointer Index
Page Directory Index
Offset inside Page
Final Page
Phy Mem
PD Table
PDE
PDP Table
PDPE
PML4 Table
PML4E
PML4 pointer from PASID Entry

Walk with 1GB Page

For the support for 1GB page size, the following mechanism is needed.



Global GTT

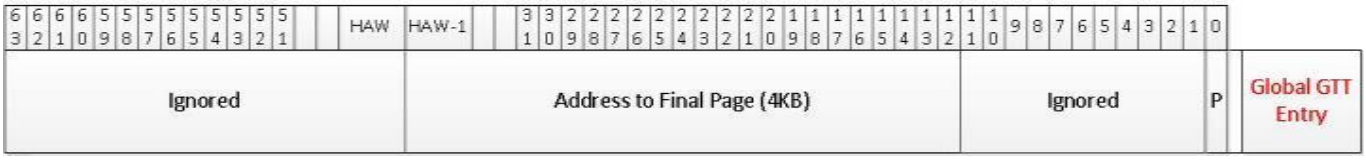
The Global GTT mechanism has the distinction of page table entry. Aperture and display will still use the global GTT even if GT core is mapped via per-process GTT.

The PTE format is updated to match per process GTT definitions and GSM is now expanded in size (2MB=>8MB) to cover for the entire 4GB (32b virtual addressing) space. Each entry corresponding to a 4KB page with 2^20 entries in GSM (each with 8B content)

For "*MI_update_GTT*", the page address provided 31:12 need to be shifted down to 22:3 for the correct QW position within the GGTT.

Page Table Entry

The following page table entry will be used for Global GTT:



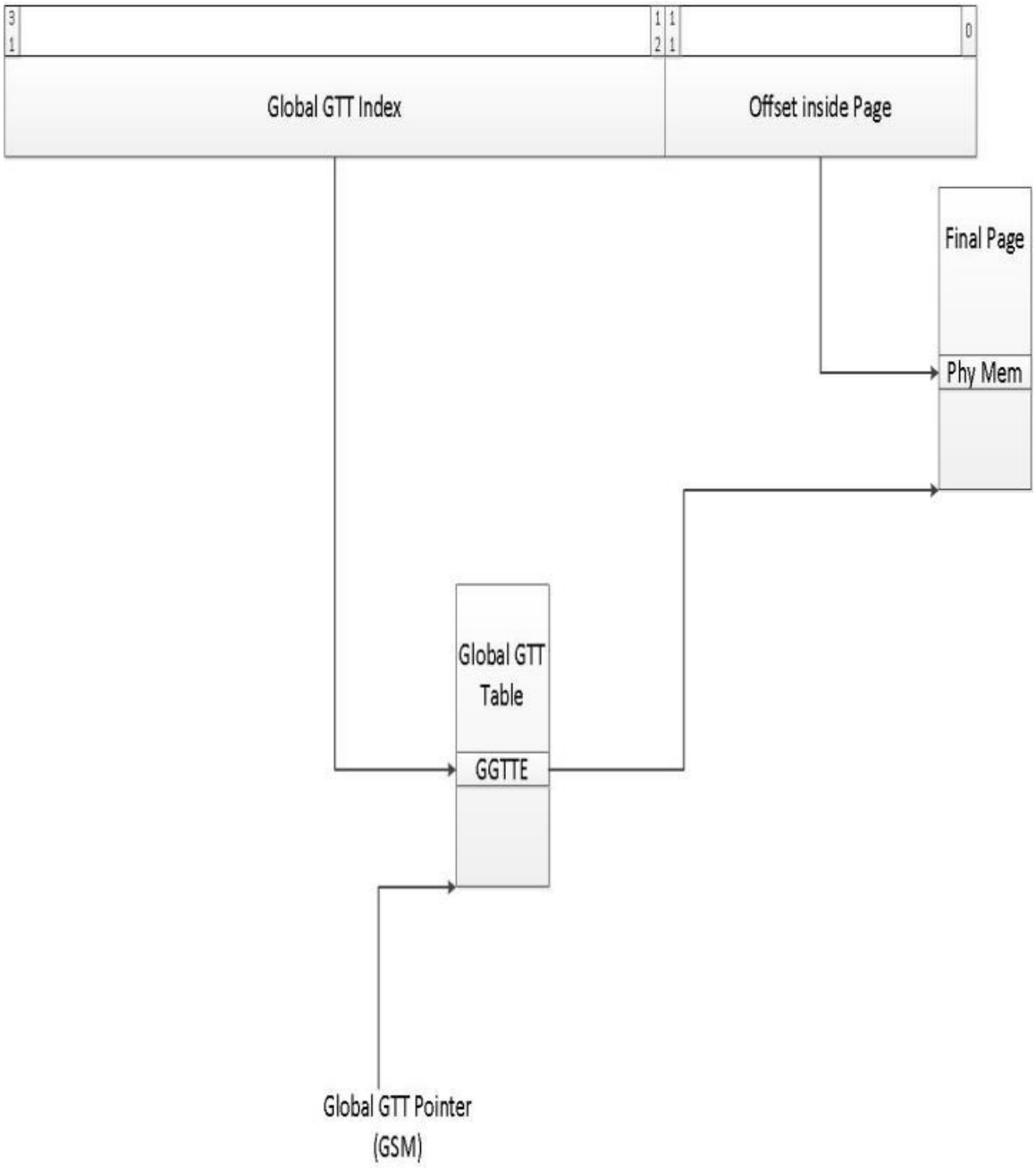
Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	Address	Physical address of 4KB memory page referenced by this entry.
11:1	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
0	Present	When set to 1, indicates that this Page Table Entry is Valid, and the corresponding page is Present in physical memory

* HAW = 39 for client, and 46 for server.

The GPU accesses GGTT table entries as uncacheable.

Page Walk

The global GTT page walk has a difference and that would be that each entry is 8B (instead of 4B) hence the entry selection needs to be updated once the corresponding Page Table miss read is returned.



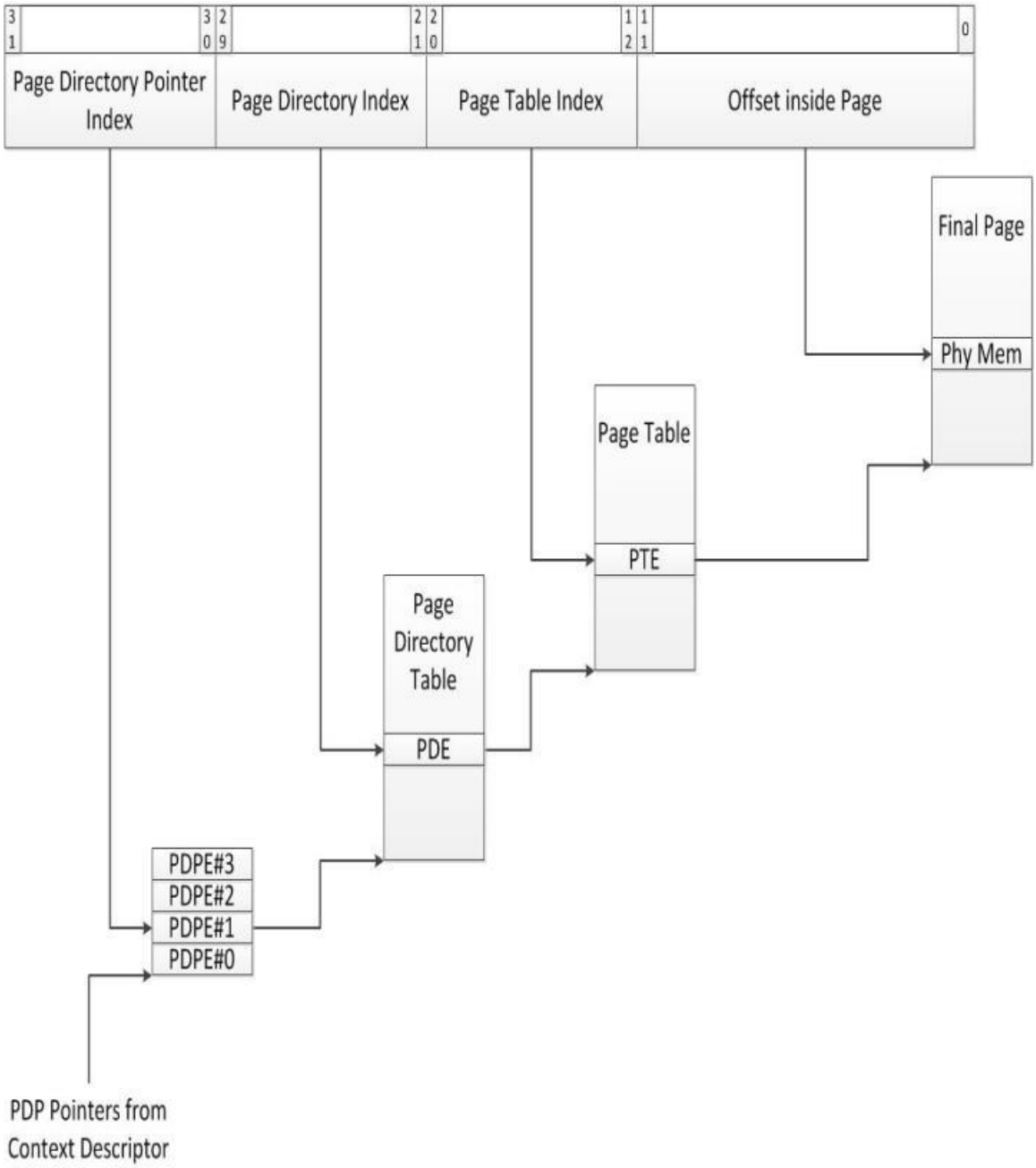
Legacy mode with 32b VA

Page walker is capable supporting 32b VA address with optimized page tables, this is to keep the walk to a single level.

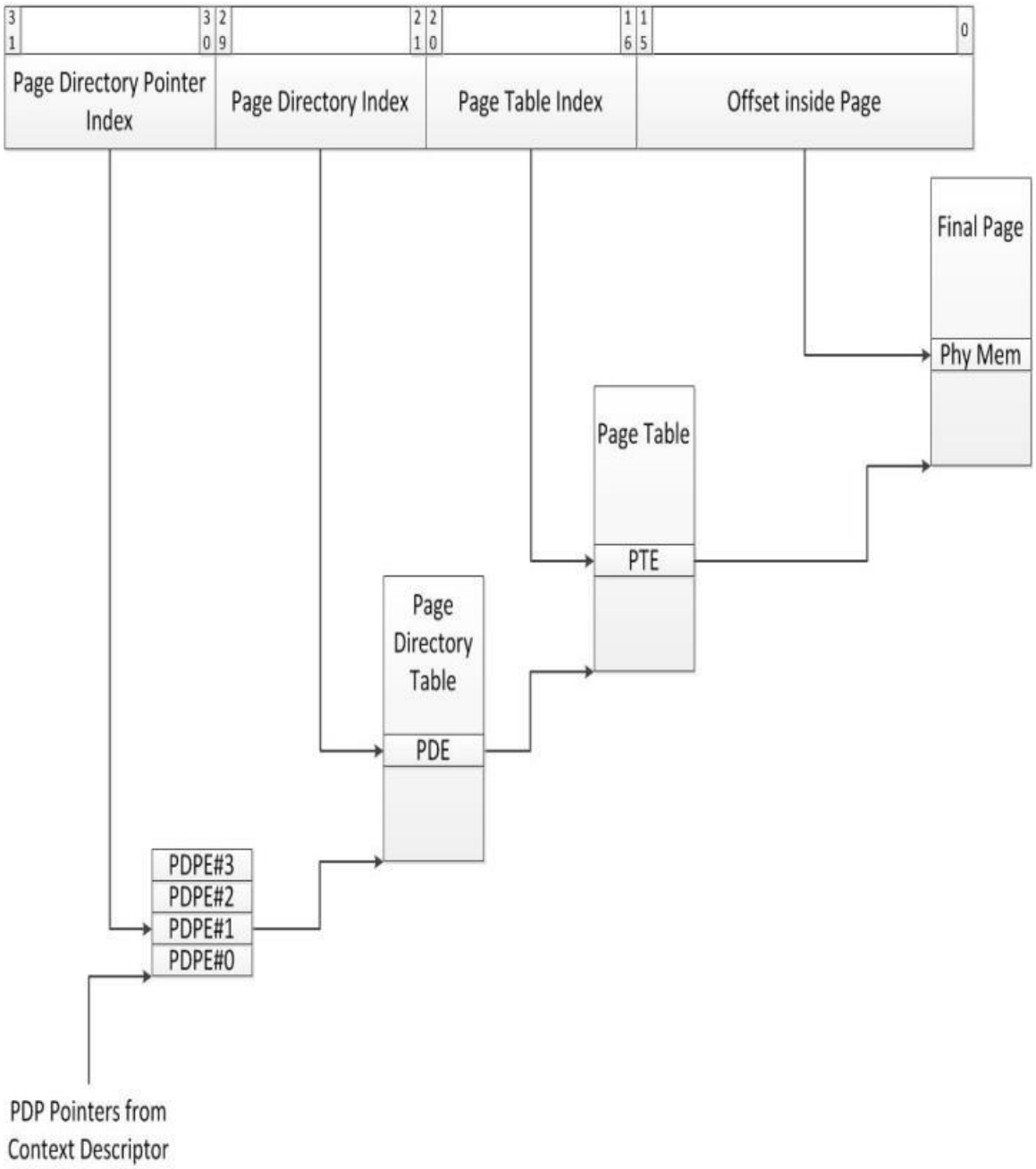
Page Walk in Legacy mode with 32b VA

For page walk in legacy mode with 48b VA, we need 2 levels. The walk will start with a PDP pointer provided by the context descriptor and uses the GraphicsAddress as an index to consecutive levels of page tables. Hardware implements 16KB intermediate caches to limit the page walk needed to a single level to have the same sensitivity to latency as previous generations.

The following diagram shows the page walk that is needed for a 4KB page.

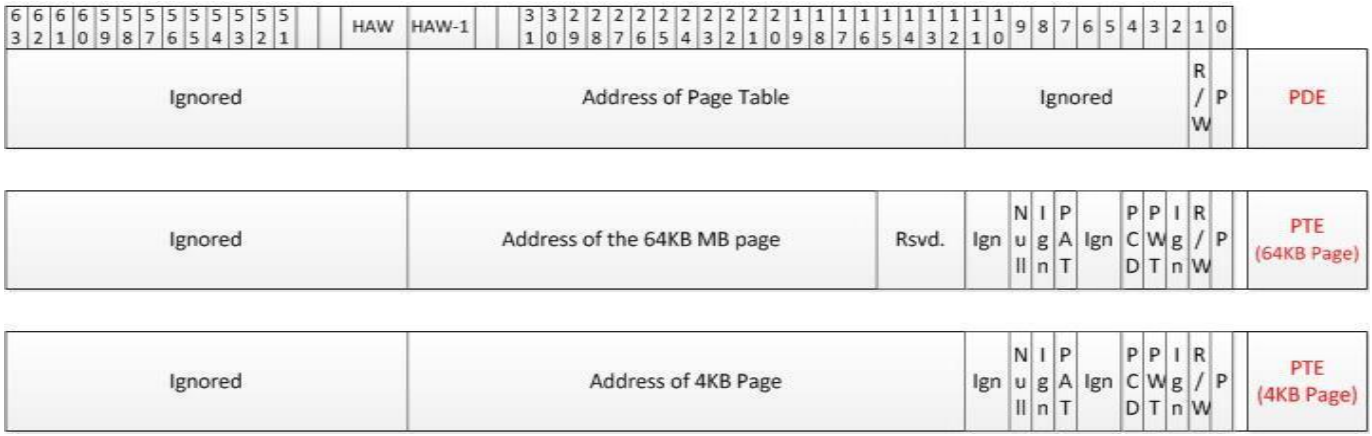

31
30
29
21
20
12
11
0
Page Directory Pointer Index
Page Directory Index
Page Table Index
Offset inside Page
Final Page
Phy Mem
Page Table
PTE
Page Directory Table
PDE
PDPE#3
PDPE#2
PDPE#1
PDPE#0
PDP Pointers from Context Descriptor

Walk with 64KB Page

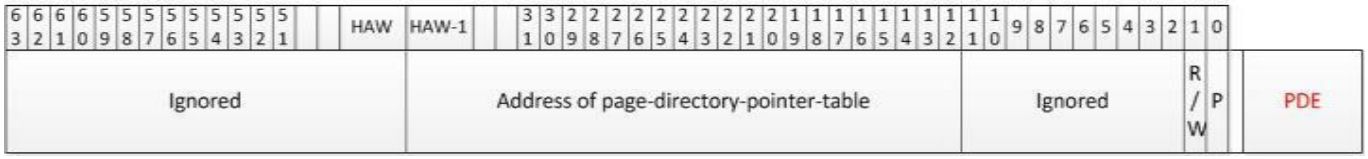


Page Table Entry (PTE) Formats

Page Table Entry formats for 32b VA use the following format:



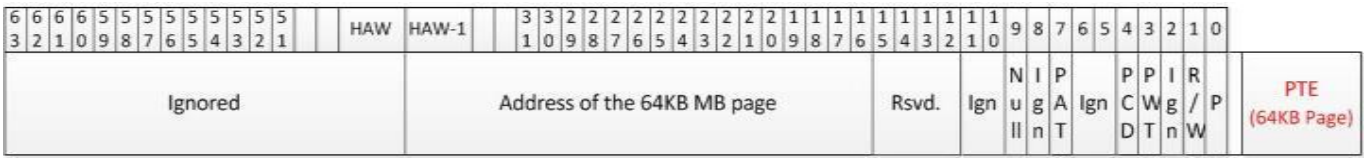
PDE for Page Table



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests targeted to the memory range pointed by this PDE. In Legacy mode with 32b VA, R/W bits from PDE are not used.
0	P: Present	PD Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

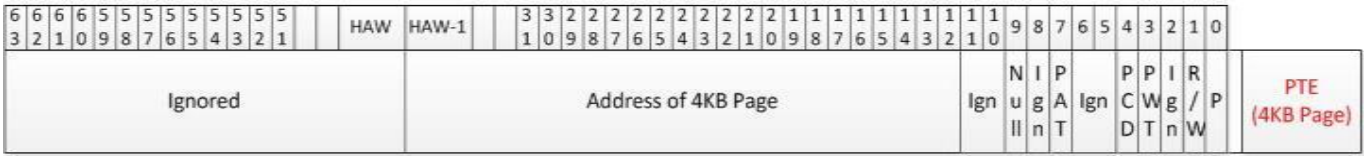
PTE: Page Table Entry for 64KB Page



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
15:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 4KB Page



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of a 4KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	This bit must be "1" to point to a valid Page.

* HAW = 39 for client, and 46 for server.

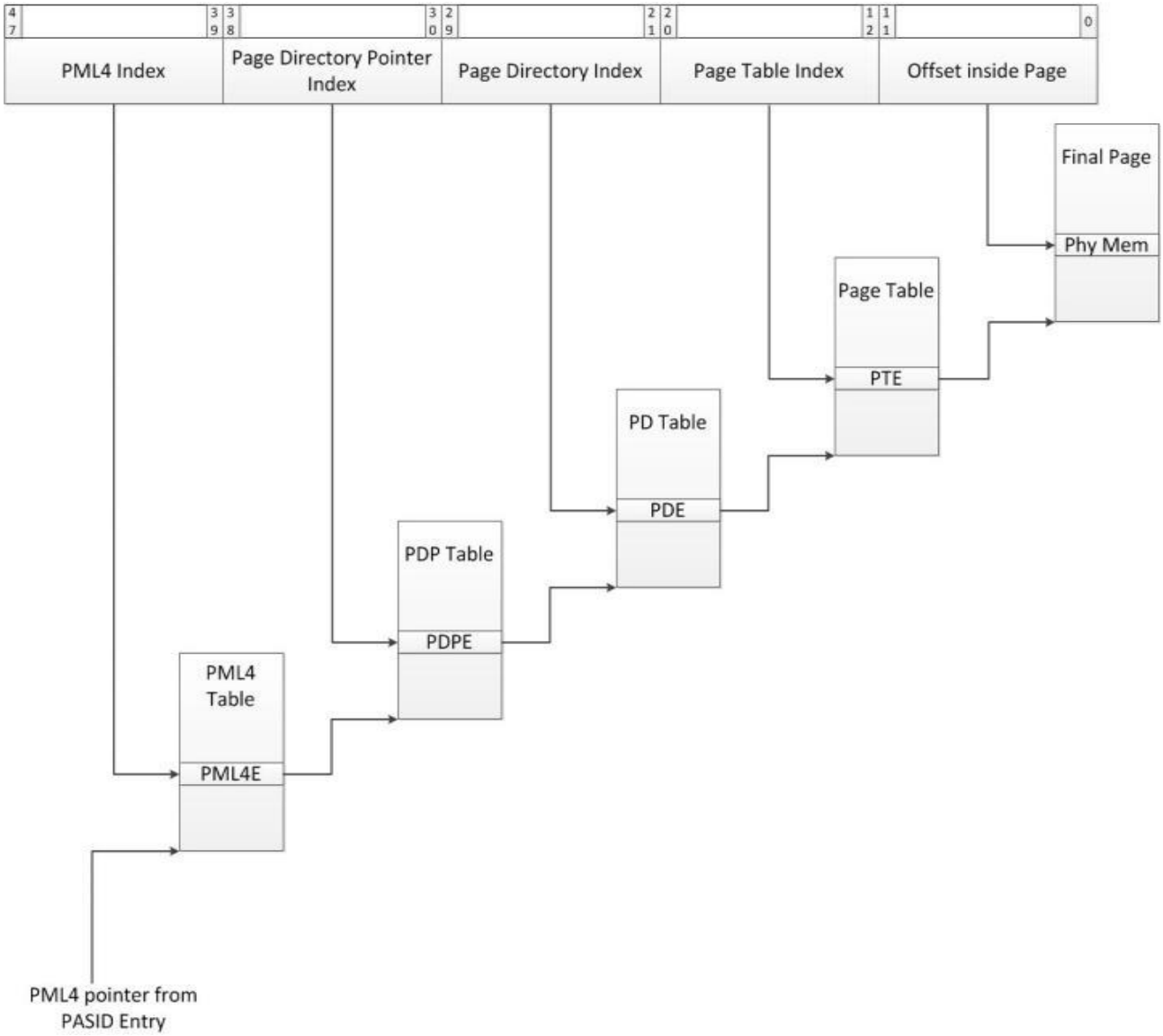
Legacy mode with 48b VA

Legacy mode with 48b VA enables larger virtual space while keeping the page walk compatible with IA32e.

Page Walk in Legacy 48b Mode

For page walk in advanced mode with 48b VA, we need 4 levels. The walk will start with a PML4 table pointer extracted from PASID entry and uses the 48b VA as index to consecutive levels of page tables.

The following diagram shows the page walk that is needed for a 4KB page.



64bit (48b canonical) address requires 4-levels of page table format where the context carries a pointer to highest level page table (PML4 pointer) via PASID. The rest of the walk is normal page walk thru various levels.

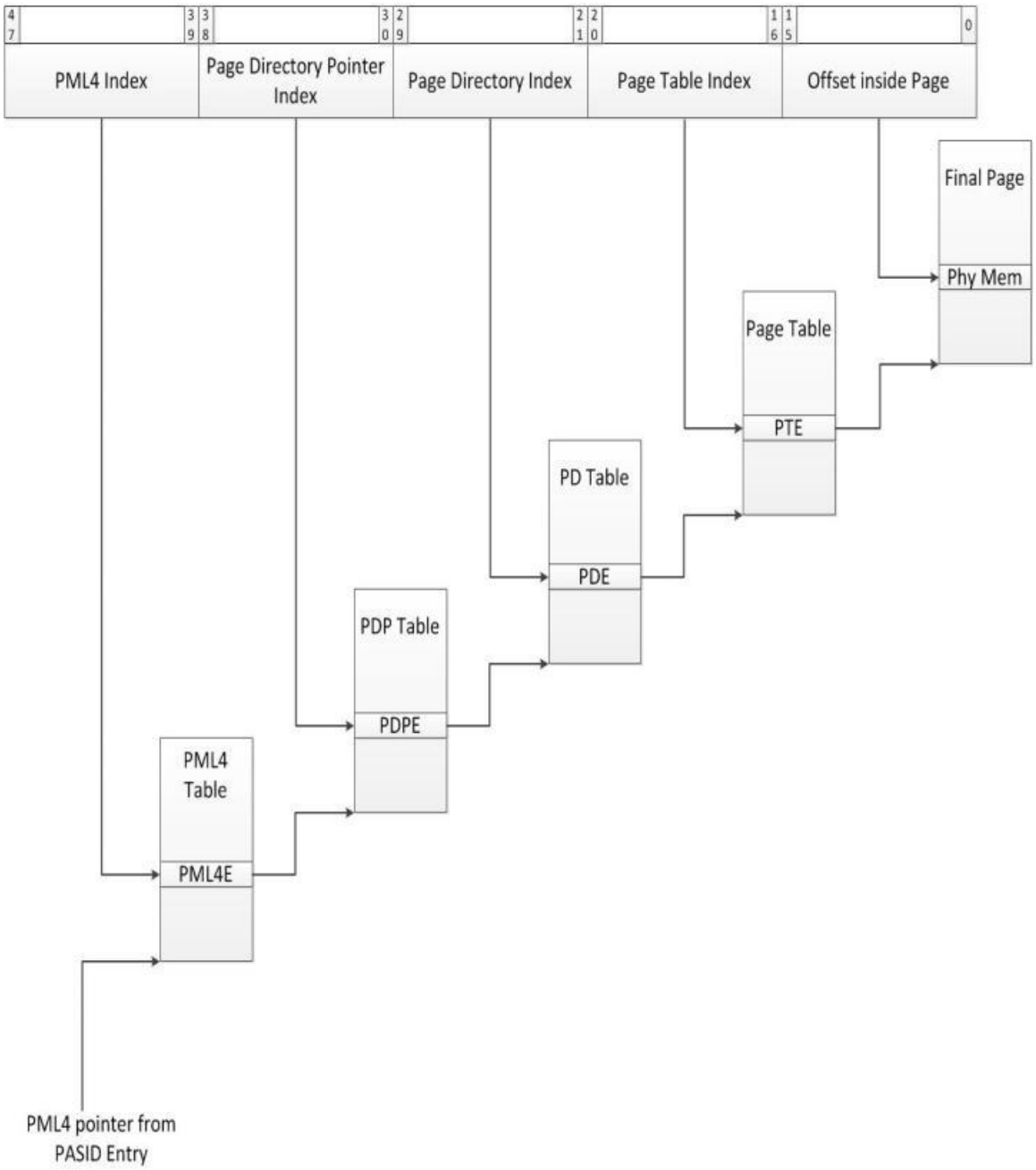
To repurpose the caches the following mechanism will be used:

- 3d: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache
- Media: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache
- VEBOX, Blitter: each with a 4KB acting as PML4, PDP, PD cache.

Note: design can section the 512 entries within 4KB to separate areas for PML4, PDP and PD.

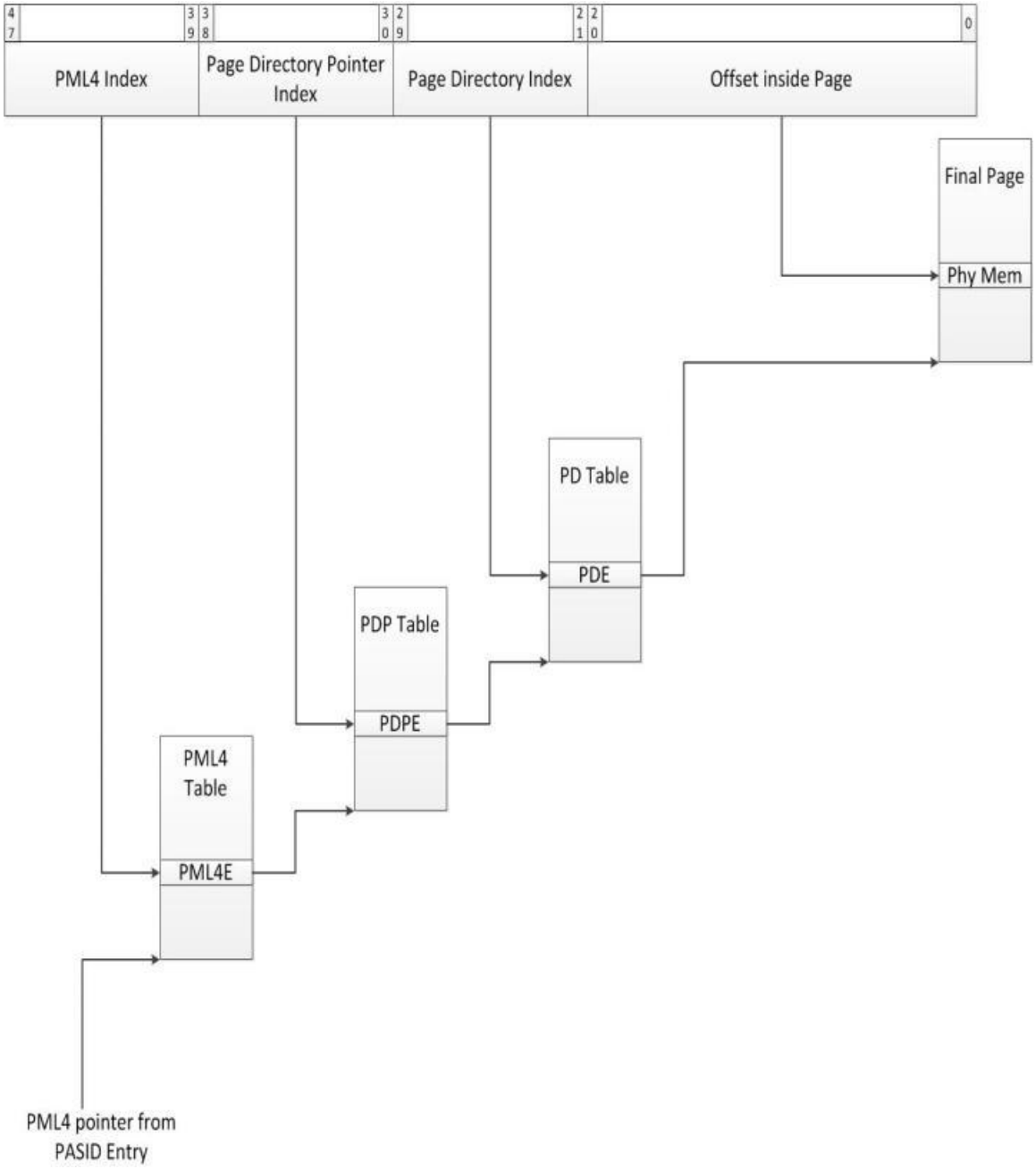
Walk with 64KB Page

64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In page table every 16th entry (PTE#0, PTE#16, PTE#32….PTE#496) should be used to index. This is calculated using address [20:16]& “0000”. Note that hardware should not make any assumptions for any other PTEs. 64K paging in the PTE is indicated by [11] of PDE. When PDE[11] = '1, every 16th PTE entry is read (by masking Adr[15:12] bits).


47
39
38
30
29
21
20
16
15
0
PML4 Index
Page Directory Pointer Index
Page Directory Index
Page Table Index
Offset inside Page
Final Page
Phy Mem
Page Table
PTE
PD Table
PDE
PDP Table
PDPE
PML4 Table
PML4E
PML4 pointer from PASID Entry

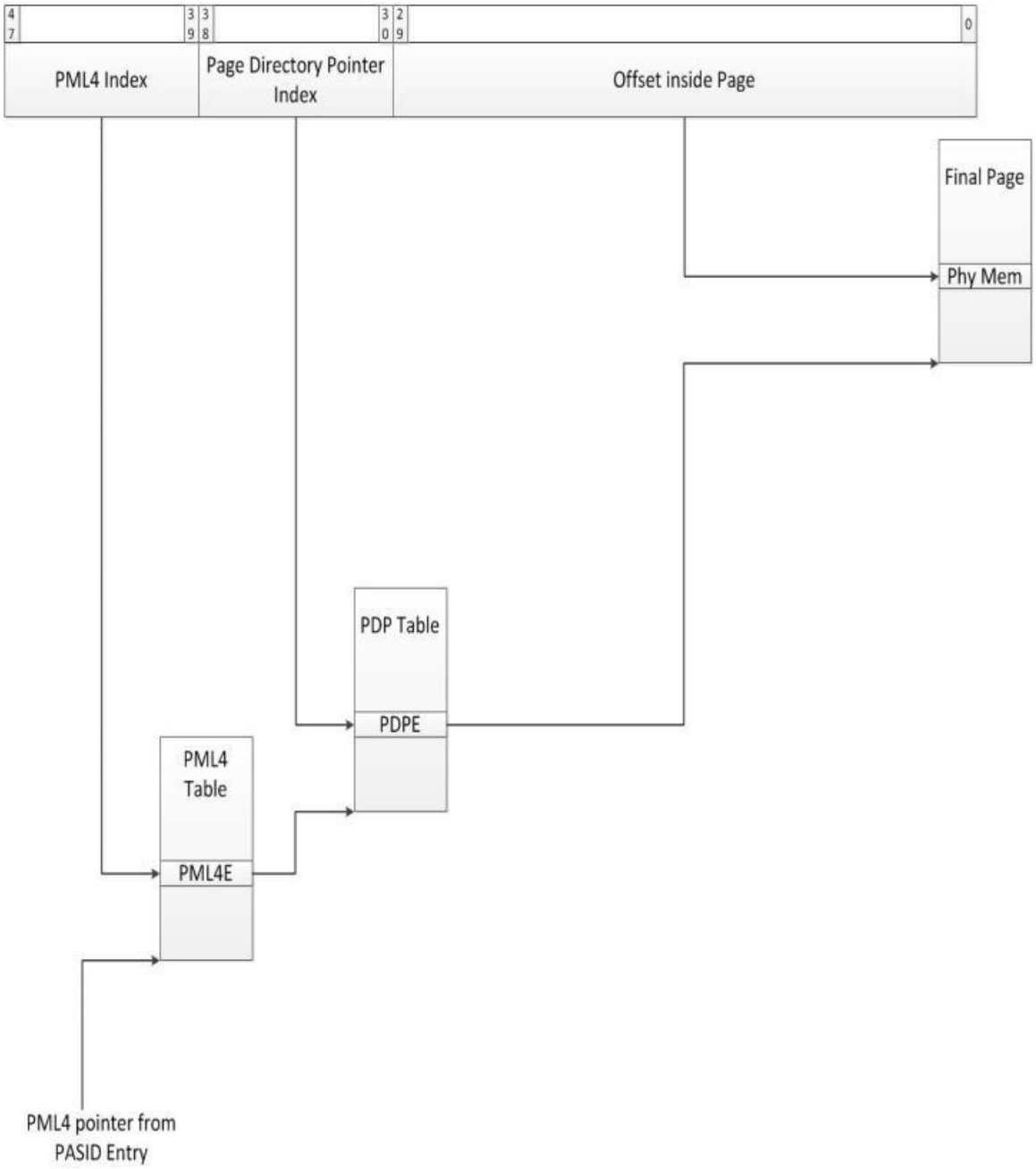
Walk with 2MB Page

With the 2MB Page walk, last level of the page walk is skipped where the PD entry points to the final page.



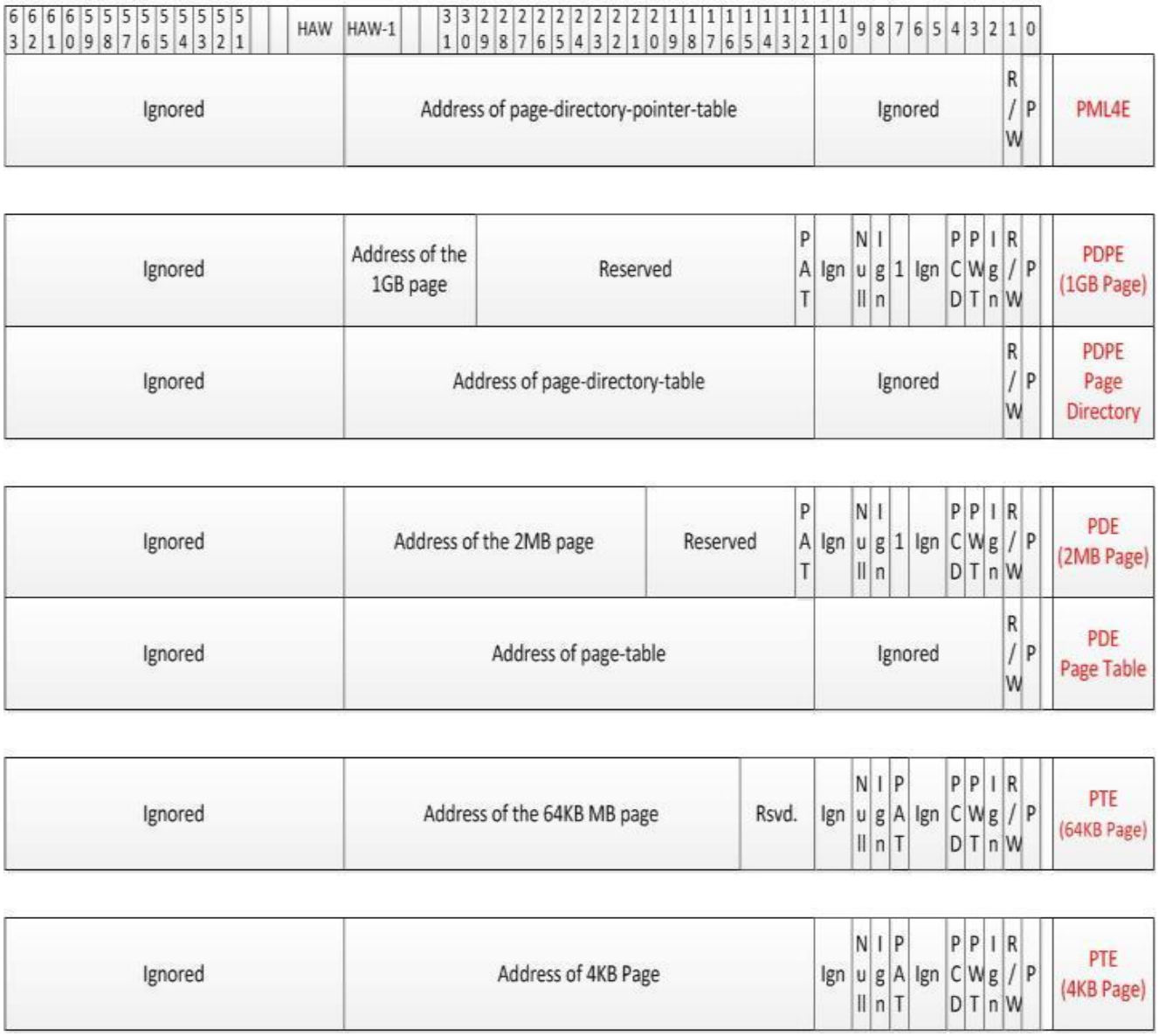
Walk with 1GB Page

For the support for 1GB page size, the following mechanism is needed.



Page Tables Entry PTE Formats

Page Table Entry (PTE) formats will follow the IA32e layout as given below:

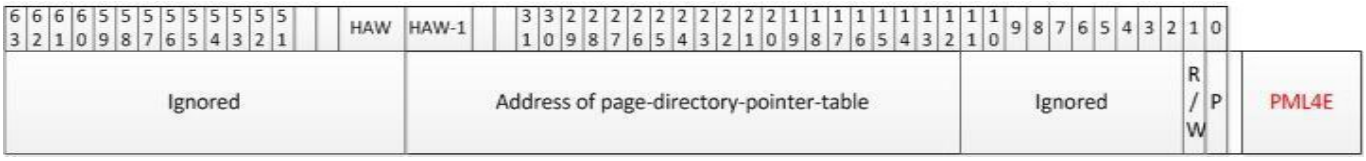


Each table entry is further broken down along with the required functions. GFX has a 4 level page table which is pointed out by context descriptor starting with the PML4. The next levels have slightly different formats depending on the size of the page supported. 1GB and 2MB page formats are required for support.

Pointer to PML4 table

In legacy mode, pointer to PML4 table is provided via the context descriptor.

PML4E: Pointer to PDP Table



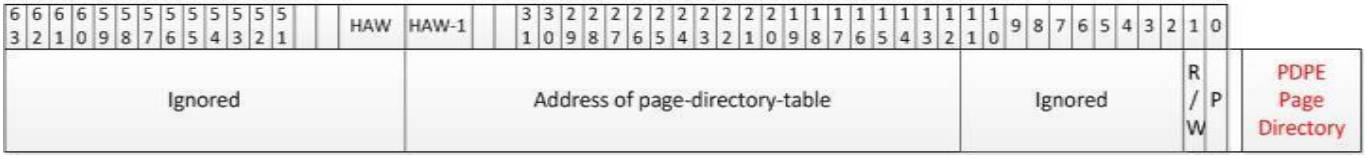
Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PML4 Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDPE: Pointer to PD Table

PDP entry is used to locate the page directory. IA32e supports 1GB pages, the PDPE has a mechanism to identify a way to say whether this PDPE represents a pointer to page directory or to a contiguous 1GB physical memory.

PDPE for PD



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. Access rights are described later. *GPU does not support Supervisor mode contexts.*
0	P: Present	PDP Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDPE for 1GB Page

Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):30	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
29:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	This bit must be "1" to point to a valid Page.

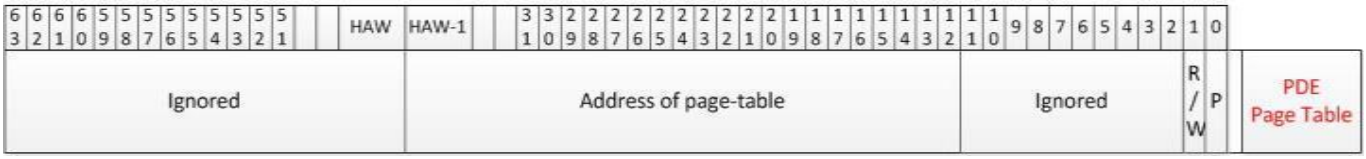
* HAW = 39 for client, and 46 for server.

PD: Pointer to Page Table

This section describes the following:

- PDE for Page Table
- PDE for 2 MB Page

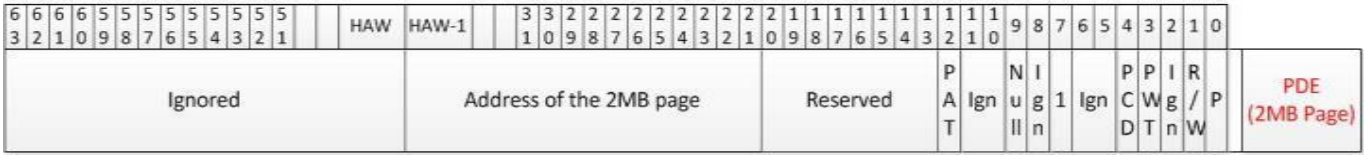
PDE for Page Table



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page- table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PDP Entry is present. The value must be "1" to point to a page directory pointer table.

* HAW = 39 for client, and 46 for server.

PDE for 2MB Page

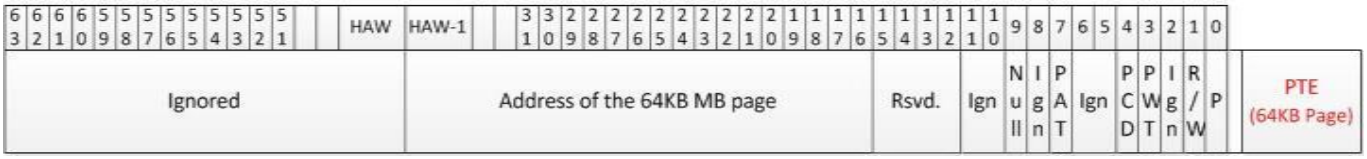


Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):21	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
20:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights.

Bits	Field	Description
		GPU does not support Supervisor mode contexts.
0	P: Present	It must be "1" to point to a 1GB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 64KB Page

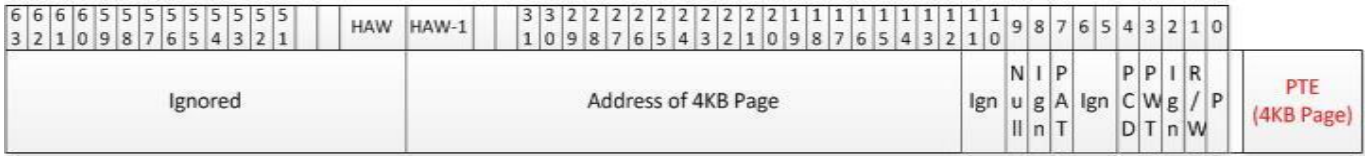


Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
15:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.

Bits	Field	Description
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 4KB Page



Bits	Field	Description
63:HAW*	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
(HAW-1):12	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11:10	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
9	N: Null	For Tile-Resources, private PPGTT tables enables for driver to merge Null Page information to primary (1st Level) translation tables. If Null=1, the h/w will avoid the memory access and return all zero's for the read access with a null completion, write accesses are dropped.
8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6:5	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.

Bits	Field	Description
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 4KB Page.

* HAW = 39 for client, and 46 for server.

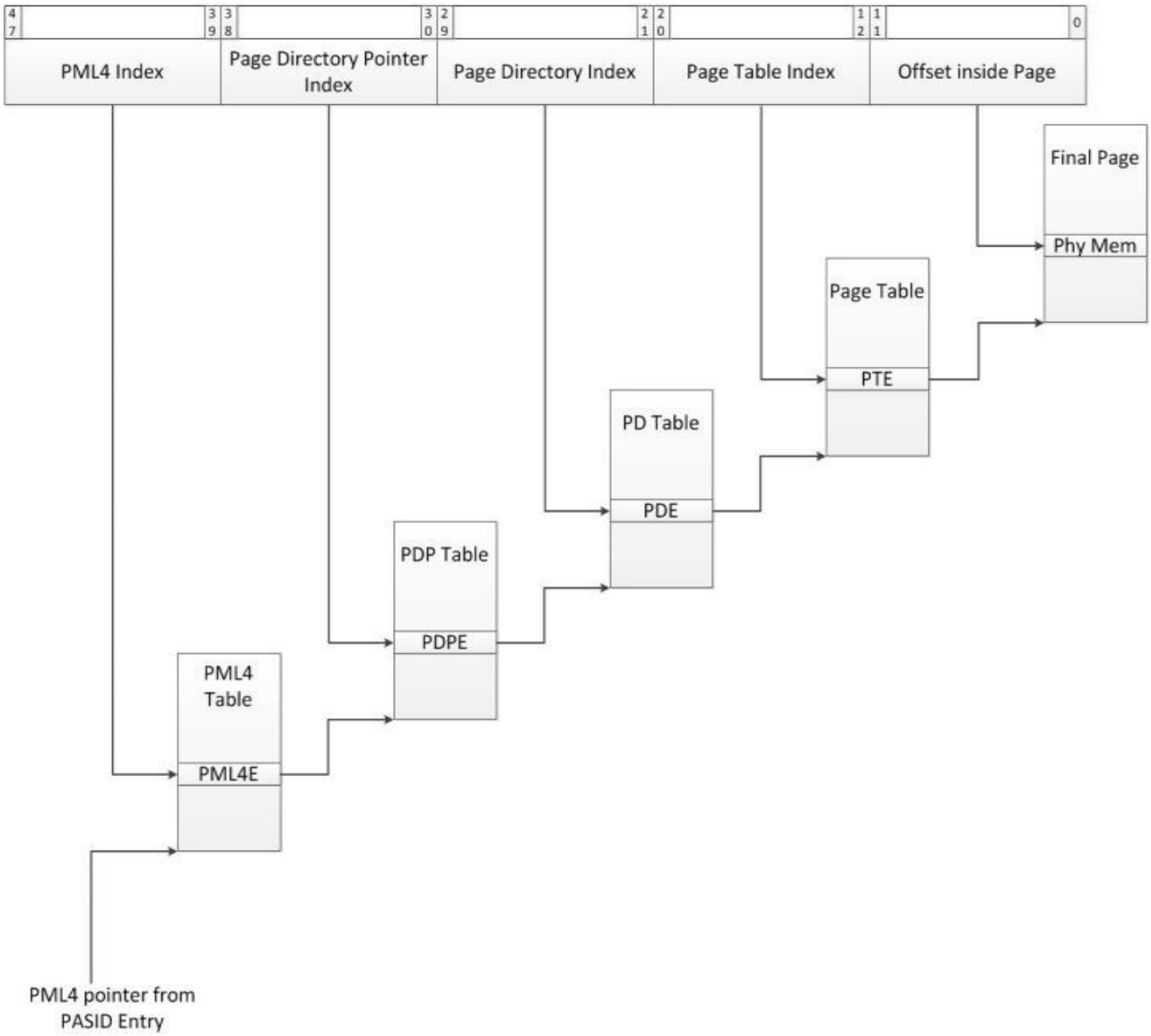
Advanced mode with 48b VA and IA32e Support

In advanced mode, per process GTT mechanism supports IA32e compatible page tables. Paging mechanism is controlled via IOMMU which shall be owned by OS or GFX driver (not both at the same time).

Page Walk in Advanced Mode

For page walk in advanced mode with 48b VA, we need 4 levels. The walk will start with a PML4 table pointer extracted from PASID entry and uses the 48b VA as index to consecutive levels of page tables.

The following diagram shows the page walk that is needed for a 4KB page.



64bit (48b canonical) address requires 4-levels of page table format where the context carries a pointer to highest level page table (PML4 pointer) via PASID. The rest of the walk is normal page walk thru various levels.

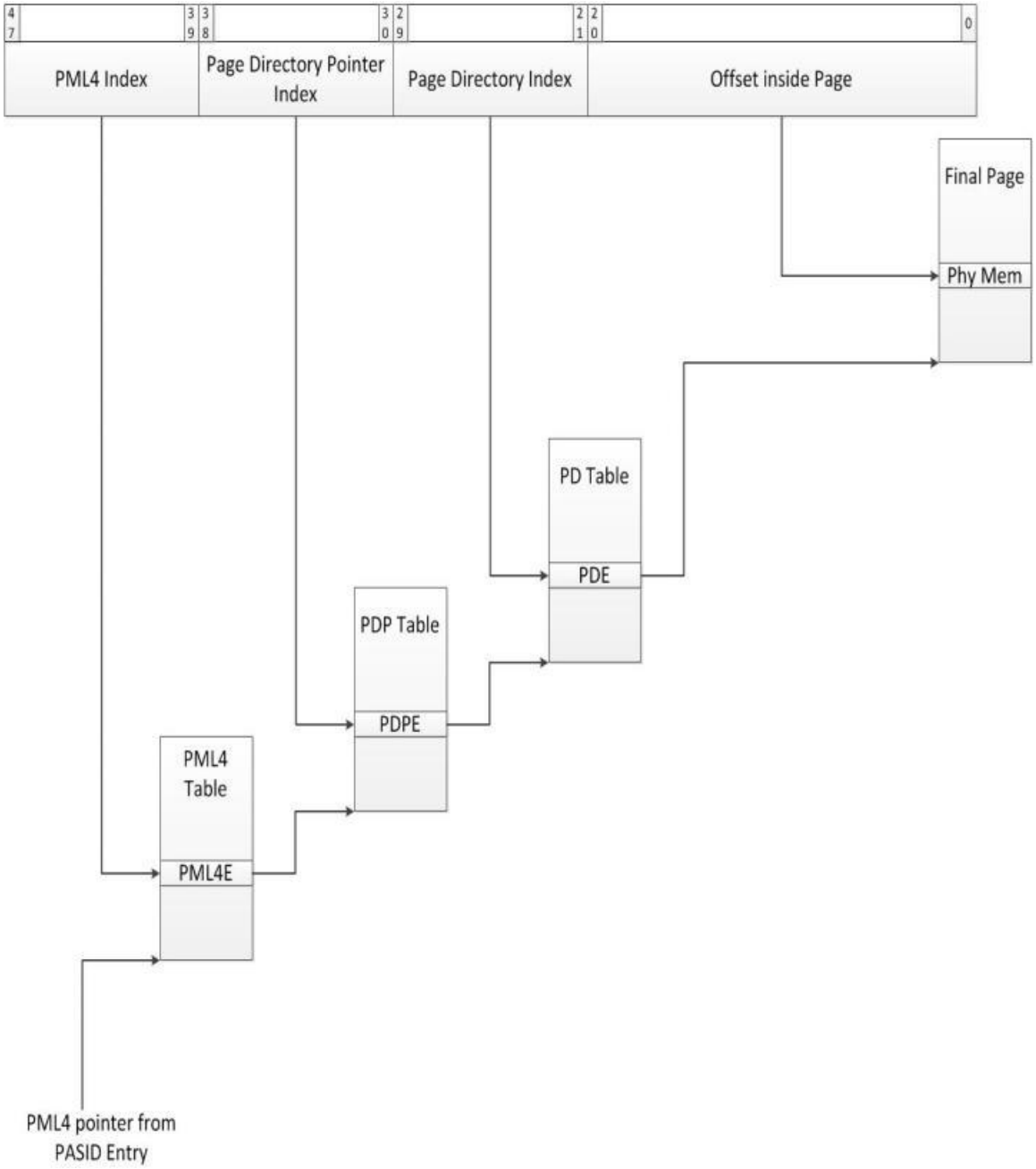
To repurpose the caches the following mechanism will be used:

- 3d: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache
- Media: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache
- VEBOX, Blitter: each with a 4KB acting as PML4, PDP, PD cache.

Note: design can section the 512 entries within 4KB to separate areas for PML4, PDP and PD.

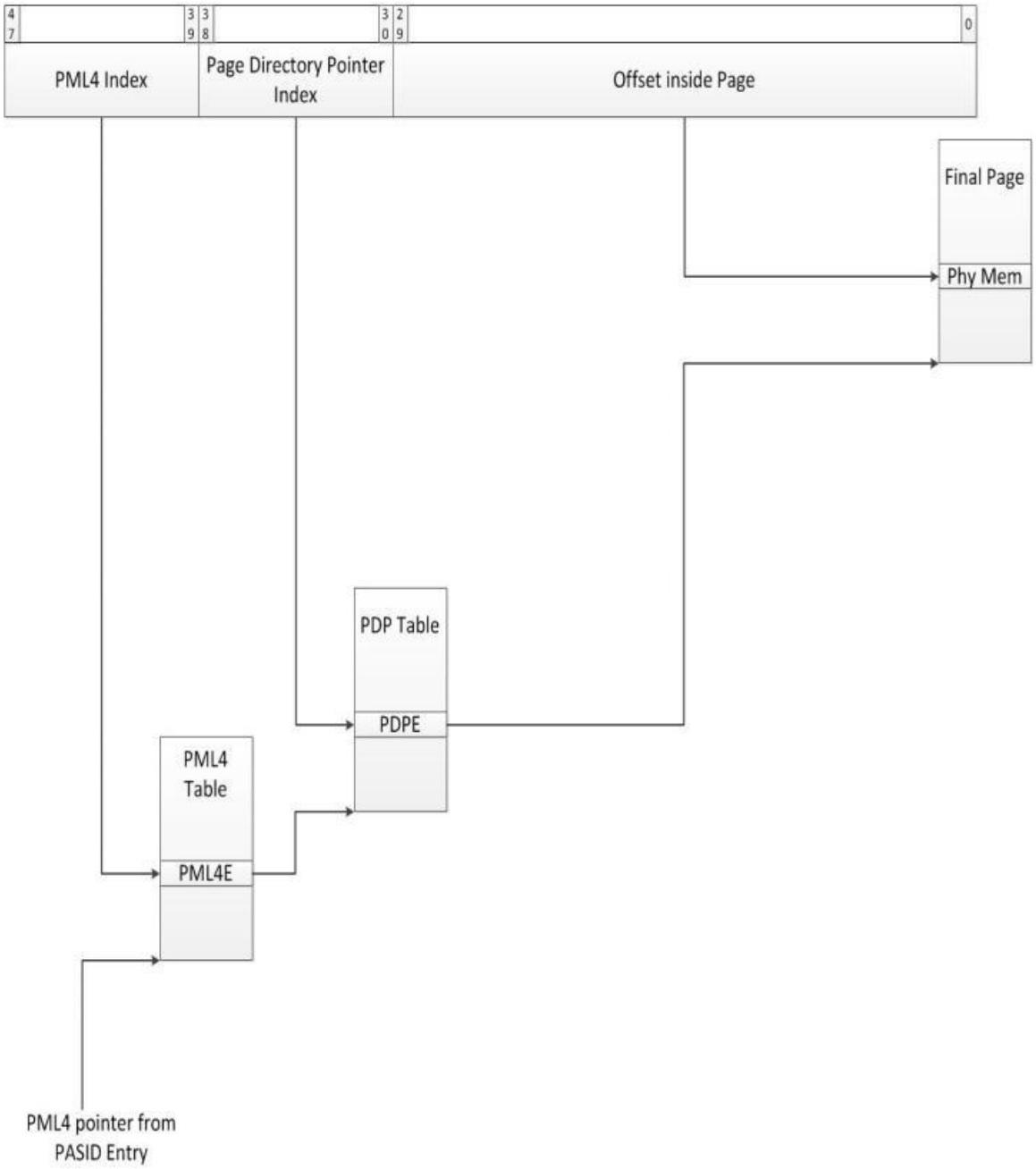
Walk with 2MB Page

With the 2MB Page walk, last level of the page walk is skipped where the PD entry points to the final page.



Walk with 1GB Page

For the support for 1GB page size, the following mechanism is needed.



Page Tables Entry (PTE) Formats

Page Table Entry (PTE) formats will follow the IA32e layout as given below:

63	62:61	60:59	58:57	56:55	54:53	52:51		HAW	HAW-1		31:30	29:28	27:26	25:24	23:22	21:20	19:18	17:16	15:14	13:12	11:10	9	8	7	6	5	4	3	2	1	0	
XD	Ignored							Rsvd.		Address of page-directory-pointer-table												Ign	EA	Ign	Rsvd	Ign	A	PCD	PWT	U/S	R/W	P	PML4E

XD	Ignored	Rsvd.	Address of the 1GB page	Reserved	PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P	PDPE (1GB Page)
XD	Ignored	Rsvd.	Address of page-directory-table			Ign	EA	Ign	0	Ign	A	PCD	PWT	U/S	R/W	P		PDPE Page Directory

XD	Ignored	Rsvd.	Address of the 2MB page	Reserved	PAT	Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P	PDE (2MB Page)
XD	Ignored	Rsvd.	Address of page-table			IPS	EA	Ign	0	Ign	A	PCD	PWT	U/S	R/W	P		PDE Page Table

XD	Ignored	Rsvd.	Address of the 64KB MB page	Rsvd.	Ign	EA	Ign	G	PAT	D	A	PCD	PWT	U/S	R/W	P	PTE (64KB Page)
XD	Ignored	Rsvd.	Address of 4KB Page		Ign	EA	Ign	G	1	D	A	PCD	PWT	U/S	R/W	P		PTE (4KB Page)

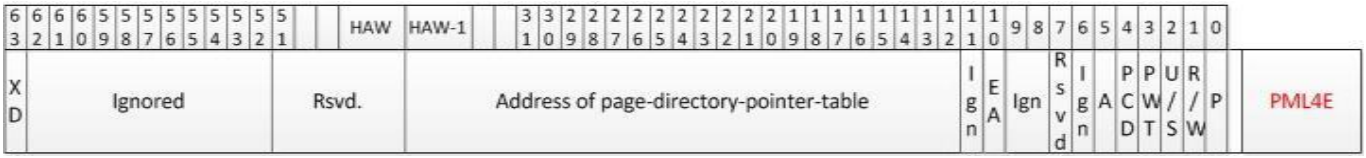
Each table entry is further broken down along with the required functions. GFX has a 4 level page table which is pointed out by context descriptor starting with the PML4. The next levels have slightly different formats depending on the size of the page supported. 1GB and 2MB page formats are required for support.

Pointer to PML4 table

Page table pointer is the starting address where the PML4 table starts. The contents of pointer will be provided by PASID table entry in case of advanced context, else it will be provided by software as part of the legacy context with 48b addressing.

Details of PASID entry is given in later sections.

PML4E: Pointer to PDP Table



Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.

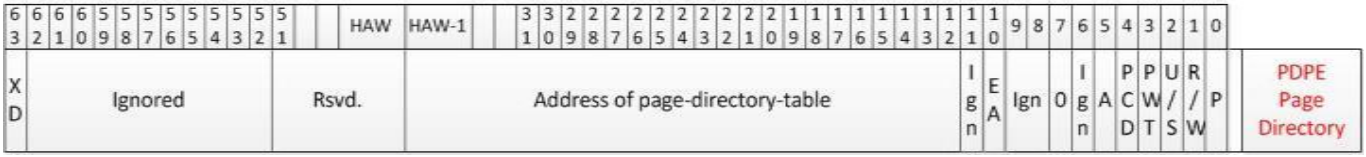
Bits	Field	Description
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PML4 Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDPE: Pointer to PD Table

PDP entry is used to locate the page directory. IA32e supports 1GB pages, the PDPE has a mechanism to identify a way to say whether this PDPE represents a pointer to page directory or to a contiguous 1GB physical memory.

PDPE for PD



Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	Ignored/Reserved	Ignored/not used by hardware

Bits	Field	Description
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PDP Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDPE for 1GB Page

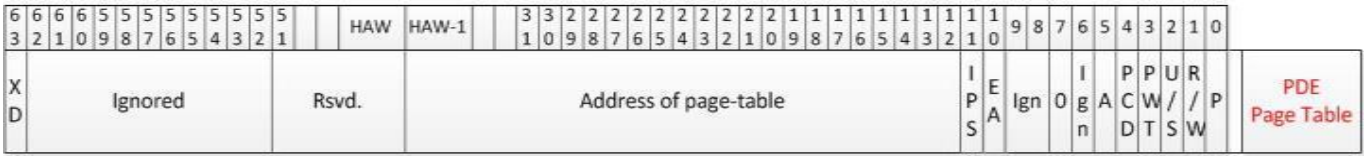
Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):30	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
29:13	Reserved	*Reserved (must return 0's)*
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	Page Size	Must be 1 to indicate 1GB page.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.

Bits	Field	Description
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	The value must be "1" to point to a 1GB Page.

* HAW = 39 for client, and 46 for server.

PD: Pointer to Page Table

PDE for Page Table

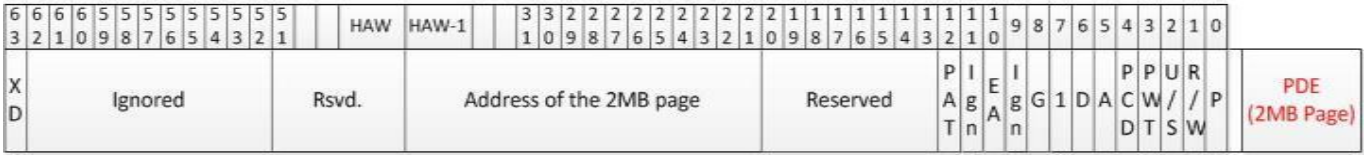


Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4-KByte aligned page table referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.

Bits	Field	Description
11	IPS	If FL64KPE=1 in the corresponding PASID entry, the page table referenced by this PD entry with IPS=1 translates into 64KB pages. If IPS=0, the page table referenced here translates into 4KB pages. If FL64KPE=0 in the corresponding PASID entry, the IPS value is ignored and the page table referenced by this entry translates into 4KB pages.
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9:8	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
7	*Reserved*	*Reserved (must return 0's)*
6	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry. GPU does not support any memory type but WB when accessing paging structures.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	PD Entry is present. It must be "1" to point to a page directory pointer table

* HAW = 39 for client, and 46 for server.

PDE for 2MB Page

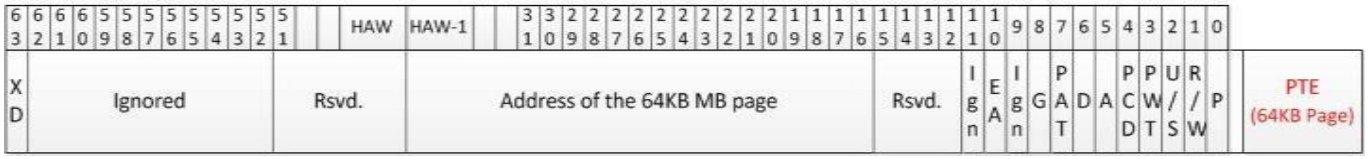


Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):21	ADDR: Address	Physical address of 1GB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
20:13	Reserved	*Reserved (must return 0's)*
12	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
11	Ignored/Reserved	Ignored/not used by hardware
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	Page Size	Must be 1 to indicate 2MB page.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.

Bits	Field	Description
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 2MB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 64KB Page

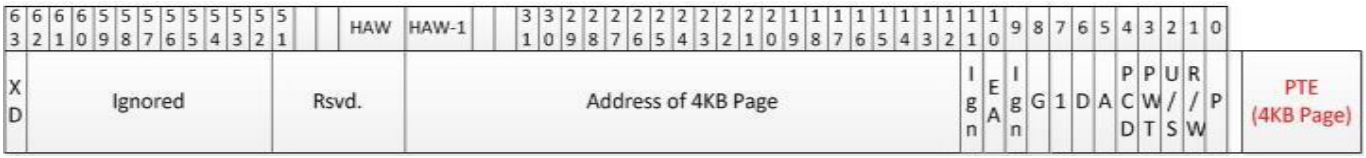


Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):16	ADDR: Address	Physical address of 64KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
15:12	*Reserved*	*Reserved (must return 0's)*
11	*Ignored/Reserved*	Ignored/not used by hardware

Bits	Field	Description
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 64KB Page.

* HAW = 39 for client, and 46 for server.

PTE: Page Table Entry for 4KB Page



Bits	Field	Description
63	XD: Execute Disable	If NXE=1 in the relevant extended-context-entry, execute permission is not granted for requests to the memory region controlled by this entry when XD=1.
62:52	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
51:HAW*	*Reserved*	*Reserved (must return 0's)*
(HAW-1):12	ADDR: Address	Physical address of 4KB memory page referenced by this entry. This field is treated as Guest Physical Address (GPA) when Nested translations are enabled (NESTE=1) in the relevant extended-context entry.
11	*Ignored/Reserved*	*Ignored/not used by hardware*
10	EA: Extended Access	Extended Access bit is added for devices to separate accesses from IA cores. If EAFE=1 in the relevant PASID-entry, this bit indicates whether this entry has been used for address translation by device. It is the devices responsibility to set this bit. If EAFE=0 in the relevant PASID-entry, this bit is ignored. This bit applies to GPU Only.
9	*Ignored*	*Ignored (h/w does not care about values behind ignored registers)*
8	G: Global	If PGE=1 in the corresponding context table entry, this field can be set by s/w to indicate that the memory region pointed by this entry can be considered global *Global paging is not used by GPU.*
7	PAT: Page Attribute	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.
6	D: Dirty	D-bit needs to be managed by h/w as the table entry is accessed with a successful write transaction. See later sections for A/D-bit management.
5	A: Accessed	A-bit needs to be managed as the table entry being accessed. Hardware needs to set this bit for the first access to the region defined with this page table entry. See later sections for A/D-bit management.
4	PCD: Page level cache disable	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory-pointer table referenced by this entry.

Bits	Field	Description
3	PWT: Page level Write-through	For devices operating in the processor coherency domain, this field indirectly determines the memory type used to access the page directory- pointer table referenced by this entry.
2	U/S: User/Supervisor	User vs supervisor access rights. If 0, requests with user-level privilege are not allowed to the memory region controlled by this entry. See section for access rights. *GPU does not support Supervisor mode contexts.*
1	R/W: Read/Write	Write permission rights. If 0, write permission not granted for requests with user-level privilege (*and requests with supervisor-level privilege, if WPE=1 in the relevant extended-context-entry*) to the memory region controlled by this entry. See a later section for access rights. *GPU does not support Supervisor mode contexts.*
0	P: Present	It must be "1" to point to a 4KB Page.

* HAW = 39 for client, and 46 for server.

GTT Cache

Processor graphics page walker implements a GTT cache which holds the remaining entries that are read as a cacheline but not used for the immediate page walk. This is only applicable in case of leaf walks and not including the 2MB/1GB page sizes. When SW enables the use of 2MB/1GB page sizes, it must disable the GTT cache.

When running advanced context, GTT cache must be disabled. This is needed as the GTT cache is not snooped, and no A/D bit updates are done on the PTEs that are cached in the GTT Cache.

GFX Page Walker (GAM)

GPU supports various engines behind the same page walker. These streams/contexts are identified Client level IDs which are carried via the arbitration pipeline. Page walker using look-up tables does the correct selection for the page tables in case of concurrent context are running at the same time.

There are two different types of page table types:

Global graphics translation table (GGTT) is a single common translation table used for all processes. There can be many Per-process graphics translation table (PPGTT). This requires an additional lookup for translation.

Virtual Memory Structure	Memory Location
Global (GGTT)	GSM Only
Per-Process (PPGTT) – private	2 to4-level, Page Tables anywhere
Per-Process (IA32e) – shared	4 levels, Page Tables anywhere

IA32e compatible PPGTT is added enable SVM (shared virtual memory) functions.

Context Definition for GFX Page Walker

Page Walker blocks need details about the context to decide on what type of page tables are used, what the error handling cases are, and many other details to operate. The information is passed to Page Walker (GAM) by the respective command streamer/DMA.

GAM needs to support the following engines:

- Render
- Media (VDBox) x2
- Blit
- VEBOX x2

The following fields are sent to GAM:

- **Context Type** (4 bits):
  - **Legacy vs Advanced Context.** Defines the context type and qualifies the rest of the fields. Same field may mean something else between the *Legacy* vs *Advanced* context. There is no restriction for what type of context can run in either combination.
    - *Requests without address-space-identifier (Legacy Context):* These are the normal memory requests from endpoint devices. These requests typically specify the type of access (read/write/atomics), targeted DMA address/size, and identity of the device originating the request.
    - *Requests with address-space-identifier (Advanced Context):* These are memory requests with added information identifying the targeted process address space from endpoint devices supporting virtual memory capabilities. Beyond attributes in normal requests, these requests specify the targeted process address space identifier (PASID), and extended attributes such as Execute-Requested (ER) flag (to indicate reads that are instruction fetches), and Privileged-mode-Requested (PR) flag (to distinguish user versus supervisor access). For details, refer to the Process Address Space ID (PASID) Capability in the PCI-Express specifications.
  - **A/D Support Enable.** Access and Dirty bits are used when OS is managing the page tables and has been added to IA32e compatible page walk. Context defines whether A/D bits need to be managed via GPU (only applicable in Advanced Context).

  - **Privileged Context Support.** Enables GPU to be able to run a privileged context which translates into page table accesses regardless of user vs supervisor privileges (only applicable in Advanced Context).
  - **32b vs 48b VA Support.** Enables 48b VA in page tables for the page walks. The rest of the HW is seamless to 32b vs 48b VA address walks, however GAM does the check and properly aligns the page walk to address bits. **Note:** Only applicable in Legacy Context. Advanced Context is always 48b.
  - **Page Fault Support Model:**
    - *Fault and Hang:* The only supported fault handling mode for legacy context and it is not applicable to advanced mode. Optionally hang can be skipped for HW to make progress.
    - *Fault and Stream (Switch if needed):* Context can survive thru a number of page faults and could be switched out by the scheduler if a certain threshold is reached.
    - *Fault and Halt:* HW detects page fault and reports to SW; the request is flagged in pending queue as "waiting for page response" and is halted until the page response is returned.
- **Function Number** – 3-bit field that defines the function number of the device. GFX device is always on BUS=0 and DEVICE=2. If we are not virtualized, our FUNCTION#=0 however if virtualized function number can be any 8 possible values (i.e. 0-7). The BUS/DEVICE/FUNCTION numbers are used for the initial walk for ROOT and CONTEXT tables.
- **PASID** – Process Address Space IDentifier: Use to identify the context that is submitted to HW. We use the PASID in many places where during the page walk (i.e. PASID table look up) or while communicating with SW on page faults. Each engine could be running an independent context with different PASID. The page walker should have a mechanism to be able to cache at least some number of PASID table entries (matching the engine count) for faster walk.
- **Context ID** (Queue ID, Bell ID) – Context ID is used to further qualify the running context beyond the PASID. PASID is given per process, and same process may allocate multiple queues to communicate with HW. The only way to further identify the process is to use an additional ID. For GFX HW Context ID could be same as the bell number assigned to it. GAM HW uses the context ID to populate the queue ID field while communicating page faults to SW.
- **Page Table Pointers** – The field could be up to 256 bits (i.e. 4x64bits) to identify the page table pointers associated with the context. For legacy 32b context, the entire 256b is valid representing the 4 PDPTR table entries. For 48b legacy context only the lower 64b is relevant pointing to base of PML4. In case of advanced context, PASID is given in the context definition.

Context Definition Delivery

Context Definition is supposed to be delivered from the corresponding command streamer to GAM and GAM has independent storage for each engine present.

Context Definition is given by *CS to GAM via a new message:

Message: “Context Available”

GAM prepares for new context, cleans up internal state and does the proper fencing. Most of these steps should have been performed when context switch request was done for the previous context, but added here for completeness.

Message: “Context Receive Ready”

GAM is ready for the context. *CS writes all new context values into the descriptor registers. To push all context descriptors CS sends the following message to GAM also indicating new context descriptor is downloaded.

Message: “Context Launched”

GAM does the context requirements and sends the following message to CS to resume its command parser.

Message: Context Confirmed

GAM should send context confirmed message only after PD restore is done. CS waiting for context confirmed message is treated as PD restore busy. Since all clients memory interface are blocked during PD restore it doesn’t make any difference if the context confirmed message is send by GAM immediately or after PD restore.

Element Descriptor Register

General Description	Element Information: The register is populated by command streamer and consumed by GAM
Register Offset	See per engine list below.

Bits	Access	Default	Field
63:32	RO	Xh	**Context ID:** Context identification number assigned to separate this context from others. Context IDs needs to be recycled in such a way that there cannot be two active contexts with the same ID. This is a unique identification number by which a context is identified and referenced.
31:12	RO	Xh	**LRCA:** Command Streamer Only
11:9	RO	Xh	**Function Number:** GFX device is considered to be on Bus0 with device number of 2. Function number is normally assigned as 000b.
8	RO	Xh	**Privileged Context / GGTT vs PPGTT mode:** Differs in legacy vs advanced context modes: **In Legacy Context:** Defines the page tables to be used. This is how page walker come to know PPGTT vs GGTT selection for the entire context. 0: Use Global GTT 1: Use Per-Process GTT **In Advanced Context:** Defines the privilege level for the context. 0: User mode context 1: Supervisor mode context
7:6	RO	Xh	**Fault Model:** 00b: Fault & Hang. 01b: Fault & Halt/Wait. Advanced Context mode Fault & Halt. If context switching is available, advanced mode is on, Fault & Halt utilizes Advanced Context capabilities as documented in "Memory Views > Page Fault Modes". 10b: Fault & Stream & Switch. Advanced context switching will be utilized if it is available. 11b: Reserved
5	RO	Xh	**Deeper IA coherency Support:** **In Advanced Context:** Defines the level of IA coherency: 0: IA coherency is provided at LLC level for all streams of GPU. 1: IA coherency is provided at L3 level for EU data accesses of GPU.

Bits	Access	Default	Field
4	RO	Xh	**A&D Support / 32&64b Address Support:** Differs in legacy vs advanced context modes: **In Legacy Context:** Defines 32b vs 64b (48b canonical) addressing format: 0: 32b addressing format. 1: 64b (48b canonical) addressing format. **In Advanced Context:** Defines A&D bit support: 0: A&D bit management in page tables is NOT supported. 1: A&D bit management in page tables is supported.
3	RO	Xh	**Context Type: Legacy vs Advanced** Defines the context type. 0: Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models, ...). Note that advanced context is not bounded to GPGPU. 1: Legacy Context: Defines the context as legacy mode. **Note:** Bits [8:4] differs in functions when legacy vs advanced context modes are selected.
2	RO	Xh	**FR:** Command streamer specific.
1	RO	Xh	**Scheduling Mode:** 0: Indicates Ring Buffer mode of scheduling. 1: Indicates execlist mode of scheduling.
0	RO	Xh	**Valid:** Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it continues but flags an error.

PDP0/PML4/PASID Descriptor Register

General Description	PDP0/PML4/PASID: The register is populated by command streamer and consumed by GAM. It contains one of the 3 values which is determined by looking at the element descriptor.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:0	RO	Xh	**PDP0/PML4/PASID:** This register can contain three values which depend on the element descriptor definition. **PASID[19:0]**: Populated in the first 20bits of the register and selected when Advanced Context flag is set. **PML4[38:12]:** Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected **PDP0[38:12]:** Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping *Note: This is a guest physical address*

PDP1 Descriptor Register

General Description	PDP1: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP1:** Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping *Note: This is a guest physical address*

PDP2 Descriptor Register

General Description	PDP2: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP2:** Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping *Note: This is a guest physical address*

PDP3 Descriptor Register

General Description	PDP3: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP3:** Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping *Note: This is a guest physical address*

List of Registers and Command Streamers

The following registers are message registers and not written directly by SW.

Engine	Offset	Description
Render	x4400h	Element Descriptor Register
	x4408h	PDP0/PML4/PASID Descriptor Register
	x4410h	PDP1 Descriptor Register
	x4418h	PDP2 Descriptor Register
	x4420h	PDP3 Descriptor Register
Media0 (VDBOX0)	x4440h	Element Descriptor Register
	x4448h	PDP0/PML4/PASID Descriptor Register
	x4450h	PDP1 Descriptor Register
	x4458h	PDP2 Descriptor Register
	x4460h	PDP3 Descriptor Register
Media1 (VDBOX1)	x4480h	Element Descriptor Register
	x4488h	PDP0/PML4/PASID Descriptor Register
	x4490h	PDP1 Descriptor Register
	x4498h	PDP2 Descriptor Register
	x44A0h	PDP3 Descriptor Register
VEBOX	x44C0h	Element Descriptor Register
	x44C8h	PDP0/PML4/PASID Descriptor Register
	x44D0h	PDP1 Descriptor Register
	x44D8h	PDP2 Descriptor Register
	x44E0h	PDP3 Descriptor Register
Blitter	x4500h	Element Descriptor Register
	x4508h	PDP0/PML4/PASID Descriptor Register
	x4510h	PDP1 Descriptor Register
	x4518h	PDP2 Descriptor Register
	x4520h	PDP3 Descriptor Register

Messages:

Message Name	Source	Destination	Category	Address	Bit	Mask Bit	Value	Description
Context Available	CS (GT)	GAM (GT)	self-clear	4004	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	CS(GT)	self-clear	3438	0	16	1	Signal ack from GAM to CS in response to Context Available message from CS to GAM.
Context Launched	CS (GT)	GAM (GT)	self-clear	4004	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	CS(GT)	self-clear	3438	1	17	1	Signal ack from GAM to CS in response to Context Launched message from CS to GAM.
Context Available	BCS (GT)	GAM (GT)	self-clear	4014	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	BCS(GT)	self-clear	23438	0	16	1	Signal ack from GAM to BCS in response to Context Available message from BCS to GAM.
Context Launched	BCS (GT)	GAM (GT)	self-clear	4014	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	BCS(GT)	self-clear	23438	1	17	1	Signal ack from GAM to BCS in response to Context Launched message from BCS to GAM.
Context Available	VECS (GT)	GAM (GT)	self-clear	4010	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	VECS(GT)	self-clear	1B438	0	16	1	Signal ack from GAM to VECS in response to Context Available message from VECS to GAM.
Context Launched	VECS (GT)	GAM (GT)	self-clear	4010	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VECS(GT)	self-clear	1B438	1	17	1	Signal ack from GAM to VECS in response to Context Launched message from VECS to GAM.
Context Available	VCS0 (GT)	GAM (GT)	self-clear	4008	0	16	1	Signal request from CS to GAM as new context is about to be submitted.

Message Name	Source	Destination	Category	Address	Bit	Mask Bit	Value	Description
Context Receive Ready	GAM (GT)	VCS0(GT)	self-clear	13438	0	16	1	Signal ack from GAM to VCS in response to Context Available message from VCS to GAM.
Context Launched	VCS0 (GT)	GAM (GT)	self-clear	4008	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VCS0(GT)	self-clear	13438	1	17	1	Signal ack from GAM to VCS in response to Context Launched message from VCS to GAM.
Context Available	VCS1 (GT)	GAM (GT)	self-clear	400C	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	VCS1(GT)	self-clear	1D438	0	16	1	Signal ack from GAM to VCS in response to Context Available message from VCS to GAM.
Context Launched	VCS1 (GT)	GAM (GT)	self-clear	400C	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VCS1(GT)	self-clear	1D438	1	17	1	Signal ack from GAM to VCS in response to Context Launched message from VCS to GAM.

Updating Page Table Pointers (aka PD Load)

In case of legacy context, driver is allowed to add/remove pages as long as it is ensured that h/w is not using these entries. Previous flows allowed for a mid-context PD load to update the PD entries and directed h/w to reload updated entries.

Pre-loading of Page Directory Entries (PD load) for 32b legacy mode is not supported. PD entries are loaded on demand when there is a miss in the PDE cache of the corresponding page walker. Any new page additions by the driver are transparent to the HW, and the new page translations will be fetched on demand. However, any removal of the pages by the driver should initiate a TLB invalidation to remove the stale entries.

Page Walker (GAM) Reset

GAM gets all the engine specific resets as well as device and bus resets to manage its internal logic domains. It is the expectation of SW when a particular GPU engine (i.e. Render, Media…) gets reset, all its related HW is cleared and comes out fresh for reprogramming. That is true for most of the logic with the exception of some shared HW blocks. The following blocks require additional steps (post-reset) from SW to further clean-up the HW:

- **Hardware TLBs**: The caching structures for the page walks are often considered shared resources. The expectation for GFX driver to clear the TLBs via "TLB Invalidate" prior to re-using the engine post reset. This is the same process that was followed on previous GPU generations.
- **Page Requests**: At the time of the reset HW may have outstanding page requests to SW for page faulted accesses. These requests could be at any level hence it is required for SW to clear these paging requests pre/post-engine reset. Engine reset ensures that no new page requests are sent from HW. Page requests could be at the "page request queue" in memory where they could be mapped to a dummy page post engine reset completion. Or they could be at the MMIO registers which will block completion of the reset; it is up to SW to service paging request interrupts without waiting for the completion of reset request.

Device reset (FLR) covers most of the page walker. However there are exceptions where all messaging towards the rest of the system (system agent) should not be impacted by it.

All external interactions and IOMMU related blocks are kept under bus (system) reset. GAM keeps the following blocks outside the device reset:

- IOMMU registers and content
- All system agent messaging structures (including translation enable flows, root pointer structures, and DMA fault reporting pieces)

An engine being reset also means the particular context that engine is running, is complete or taken out. This requires GAM to decrement the PASID_State Counter if the engine was running a PASID based (advanced) context. For FLR (device reset) similar requirement holds. In case of device reset, GAM needs to decrement all the PASID state counters that are active on the GPU before completing the sequence.

TLB Caching and Management

As compared to previous generation of TLB entry, IA32e page translation entry is quite different. At every stage of the page different bits need to be taken into account and proper treatment is required. Regardless of PPGTT vs GGTT usage, the paging entry has the same format. Linear address are translated using a hierarchy of in-memory paging structures located using the contents of CR3. IA-32e paging translates 48-bit linear addresses to 52-bit physical addresses.1 Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.IA-32e paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.

63	62	61	60	59-M	M-1	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved[2]						Address of PML4 table																	Ignored											PCD	PWT	Ign.			CR3
XD	Ignored				Rsvd.	Address of page-directory-pointer table																	Ign.									Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E: present
Ignored																																						0	PML4E: not present
XD	Ignored				Rsvd.	Address of 1GB page frame													Reserved				PAT	Ign.							G	1	D	A	PCD	PWT	U/S	R/W	1	PDPTE: 1GB page
XD	Ignored				Rsvd.	Address of page directory																	Ign.									0	Ign	A	PCD	PWT	U/S	R/W	1	PDPTE: page directory
Ignored																																						0	PDTPE: not present
XD	Ignored				Rsvd.	Address of 2MB page frame													Reserved				PAT	Ign.							G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored				Rsvd.	Address of page table																	Ign.									0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																																						0	PDE: not present
XD	Ignored				Rsvd.	Address of 4KB page frame																	Ign.									PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																																						0	PTE: not present

The following rules apply:

1. *M is an abbreviation for MAXPHYSICAL ADDRESS*
2. *Reserved fields must be "0"*
3. *Ignored field must be ignored (there could be private information)*
4. *All ignore options are part of the context entry and coming from IOMMU definition.*

TLB Caches

The caching structures are separated as following with the architectural view, this is also applicable to s/w view of these caches when it comes to invalidations.

Context Cache - CC

This is the storage for context table entry which is achieved as part of root/context table walk.

Context cache can also be invalidated with directed invalidations, where HW needs to invalidate the content of the context cache along with all low level caches.

PASID Cache - PC

This is where the HW copy of the PASID table entry is kept and it is per context. This makes it unique for every HW engine that could be running an independent context (per GAM):

- Render/GPGPU
- MFX (VDBOX) – 1
- MFX (VDBOX) – 2
- Video Enhancement (VEBOX) – 1
- Video Enhancement (VEBOX) – 2
- Blitter

The cache content is updated if the corresponding engine is running an advanced context where its page table pointers are accessible via PASID table. In case of legacy context running engine, corresponding PASID Cache entry is not valid. Recommendation is to keep ONE physical storage per engine which is filled/invalidated during the context switch time.

PASID Cache can also be invalidated with the directed invalidations along with low level caches and needs to be re-filled prior to context resuming.

Intermediate Page Walk Caches (PML4, PDP, PD) – PWC

These are the stages where intermediate page walk entries are cached to speed-up/shorten the page walk when final TLB is missed. Each level can be cached separately or along with different levels, the cacheability structures will have programmability to move the boundary of different levels to accommodate more/less on each page walk level. However as a concept, for legacy 32b addressing mode, requirement is to cache 4PDPs along with 4x4KB PDs for certain engines, at least for render and media. The others will use cache concept.

TLB – Final Page Entry

The size of the TLBs has been increased over the previous generation and should be targeting using the following list:

- L3 TLB: 768 TLB entries – This is where all HDC, I$, Constant, State, and Sampler streams are stored.
- MFX: 512 TLB entries – All Media streams (split 256/256 between two media engines).
- BLT: 32 entries.
- Z: 512 TLB entries – All depth accesses.
- C: 256 (256 TLB entries) – All color accesses.
- FF: 128 (128 TLB entries) – All FF accesses to memory.
- VLF: 32 (32 TLB entries) – Media surface.
- GAV: 64 (64 TLB entries) – Video enhancement.

All TLB entries are increased to 48b to contain larger address as well as the page attributes attached to it.

The max size of a single TLB is 256 entries, larger quantities have to be handled as set-associative storages. Set associativity will be managed by low order page bits (i.e. address#12, address#13, ...).

Both Color and Z TLBs are designed to process a single memory request per cycle. To achieve a higher throughput where concurrent Color or Z read/write's are used, following register bit needs to be enabled: mmio0x04A30h [31]

The size of the L3 TLB is also different between projects. The default TLB entry alocations are:

- (L3TLB-Gfx **640**): L3(**80**:*0-79*), DC(**100**:*80-179*), TX(**444**:*180-623*), GATR(**16**:*624-639*)
- (L3TLB-GPGPU **640**): L3(**80**:*0-79*), DC(**460**:*80-539*), TX(**100**:*540-639*)

For giving more TLB resources for both DC and TX, the following allocations are recommended.

- (L3TLB-Gfx **640**): L3(**80**:*0-79*), DC(**544**:*80-623*), TX(**544**:*80-623*), GATR(**16**:*624-639*)
- (L3TLB-GPGPU **640**): L3(**80**:*0-79*), DC(**560**:*80-639*), TX(**560**:*80-639*)

TLB Entry Content

When a page walk entry is cached (or loaded prior to context start), certain bits need to be cached as well along with the physical address bits. The treatment on these bits would be considered when a HIT vs MISS decision needs to be made during a look up.

The purpose of caching is to accelerate the paging process by caching individual translations in **translation look-aside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number:
    - The logical-AND of the R/W flags.
    - The logical-AND of the U/S flags.

- The logical-OR of the XD flags.

- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
  - The dirty flag.
  - The memory type.

PRESENT: This is the same VALID bit description we had in previous page table designs. The lack of present bit (i.e. bit[0]=0) points that rest of the information in the page table entry is being invalid. For some fault models, even NOT PRESENT entries are cached to filter further page faults (*see fault models on caching page faulting entries*). If such entry is cached, there are couple ways that it can be removed from the page tables:

1. LRA selection where the entry becomes a victim for replacement
2. Global or Selective invalidation
3. Page fault response stating the faulting page is now fixed.

R/W Privilege: Certain pages can be allocated as read-only and write operations are not allowed. To make this check work, TLB has to keep the R/W bit. This bit has no affect on read operations; however for write operation privilege needs to be checked. If there is mis-match, the result of the TLB look-up should be a MISS. This does not mean a page fault immediately; the walk has to be re-done as for any TLB MISS result. There are cases OS may change page table privileges without invalidating pages in TLB (*note: all downgrades result in invalidation of the TLB, however upgrades can be done silently hence re-walk is required*). In case where the TLB Miss is due to privilege mis-match, the existing entry from TLB has to be invalidated and page walk will bring in the most up-to-date copy from memory.

The R/W privilege on final frame is generated as a logical-AND process of all upper page walks pointing to this location.

User vs Supervisor Privilege: The GPU typically operates in user mode when it comes to page tables. So the GTT walk can be treated as faulted when GPU encounters a page with supervisor privileges and the context is marked as user mode. The faulted entry can be cached back into TLB with "P" bit off indicating a faulted entry. However the page fault report should carry the correct reason why h/w detected the fault in the first place which was the user vs supervisor privilege. There is an option in context header to define the context as supervisor, than it legal to access supervisor pages.

- This is not stored in TLB

The U/S privilege on final frame is generated as a logical-AND process of all upper page walks pointing to this location.

Accessed Bit: This where a stage of the page walk cannot be used if the accessed bit is not set for that level in the page walk. This is true for both storage into TLB as well as to make progress on the page walk. In order to achieve the process of Accessed bit, every stage of the ppGTT read is done via a new semantics between the GAM and GTI such that GTI can atomically process A-bit w/o running into access violations. The details of the semantics are defined as part of the following sections. The "A" bit does not

need to be stored as part of the TLB, just the fact that a valid page table entry is present in the TLB does mean that h/w took care off the "A" bit at the time the page was brought up to TLB. Note that TLB prefetching is disabled when A-bit management is enabled.

IA32e mode page tables can not co-exist with TLB pre-fetching due to lack of A-bit management for all entries of the line.

- This is not stored in TLB

Dirty Bit: Similar to accessed bit, dirty bit needs to be managed. It is only applicable for "write" accesses. Given there are cases where a TLB entry was acquired as part of a read operation, the presence of D-bit should be maintained with the TLB. This gives us the capability to declare a TLB miss for a write access when the D-bit is not set even though TLB has a valid translation. In such case, The TLB entry needs to invalidated and the final stage of the walk needs to be re-done to ensure most up-to-date copy of GTT entry is brought into h/w. The operation of Dirty bit update is also atomic similar to A-bit management.

Execute (XD) Bit: XD bit is also present on every stage of the walk and applicable to executable code that GT would be fetching. In the first pass, instruction cache accesses are not allowed to proceed if the corresponding page does not have the execute credentials set properly. Similar treatment of the TLB entry as privilege bits is expected. A page entry that was already cached in TLB and later accessed for instruction space will have to check the XD bit which is also stored in TLB. If mis-match, the end result is a TLB miss and walk has to be re-done replacing the different stages of the walk.

The XD privilege on final frame is generated as a logical-OR process of all upper page walks pointing to this location.

Faulted Bit: There are usage models where the faulted entries are cached in TLB. This is to filter further faults to the same page as opportunistic way to prevent fault storms. When faulted bit is set the address is included in the TLB look up but final treatment is fault filtering. The rest of the bits are used to define what would be the reason for the fault. If the look-up conflicts with the original faulted reason, a re-walk is required. As a basic case, take a read access bringing up a PTE with W-flag cleared. A subsequent write access has a conflict on privilege, and it will perform a re-walk. If the result of the re-walk is W-flag set, than TLB is upgraded and write makes progress. However if the result is still W-flag cleared, the write access will fault and TLB entry will be tagged as a faulted entry with only read-allowed. Subsequent write accesses will be filtered as fault but read accesses should cause a re-walk of the page and if successful, the TLB can be updated with PTE as valid with read-only attribute.

TLB Accessed and Dirty Flags

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag. For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor and/or GPU uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor and/or GPU sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are “sticky,” meaning that, once set, the processor and/or GPU does not clear them; only software can clear them.

A processor and/or GPU may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the GPU might not set the corresponding bit in memory on a subsequent access using an affected linear address

Accessed bit is applicable to every stage of the page walk, however the dirty bit is only applicable to final stage of the walk.

The rule states that a particular access cannot be committed until the Accessed and/or Dirty bits are not visible to page management s/w. In order for GPU to follow the rule, GTT accesses (when A/D bits are supported) are going to be done via a special cycle definition between GAM and GTI.

Updating A/D Bits

New atomic operations are added to GAM to GPU interface (GTI) to handle paging entries. GAM has to set the correct atomic opcodes based on the access type and context entry controls as well as level of access.

Requires setting for opcodes are given in the table below. The steps of operations in the atomic ALUs are given later in the document.

The Following Atomics are only applicable in GTI and used for Page Walks

R/W => Bit[0]

Extended Access required => Bit[1]

Write Protect Enable => Bit[2]

Intermediate Entry => Bit[3]

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
Atomic_Page_update_0000	1100_0000	Read Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0001	1100_0001	Write Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0000	1100_0010	Read Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,10] if not set		new_dst

Atomic_Page_update_0001	1100_0011	Write Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_0100	1100_0100	Read Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0101	1100_0101	Write Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0100	1100_0110	Read Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0101	1100_0111	Write Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,6,10] if not set		new_dst

Atomic_Page_update_0000	1100_1000	Read Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5] if not set		new_dst
Atomic_Page_update_0001	1100_1001	Write Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0000	1100_1010	Read Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0001	1100_1011	Write Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_0100	1100_1100	Read Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5] if not set		new_dst

Atomic_Page_update_0101	1100_1101	Write Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0100	1100_1110	Read Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0101	1100_1111	Write Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst

Atomic updates are only possible for cacheable memory types. There could be cases where the PTE could be in WT/WC/UC space where atomic update is not possible via WB space. Those are the cases where IA cores use bus lock to update the A/D bits in PTE.

GT core is not capable of supporting bus locks and has the following options. These options will be enabled/disabled via register space.

Option#1: Ignore the PAT/MTRR setting of the PTE and update the space as WB with atomic ops. This is the place GAM will decide to go forward with atomic updates assuming WB space works

Option#2: Once the memory type is determined and the end result of the page is WC/UC/WT space, we can not guarantee an atomic update. GAM will report an application error (catastrophic) to the scheduler and handle the case as error.

Bit	Access	Default Value	Description
1	R/W	0b	**A/D Bit Update on non-WB Space:** A/D bit updates are only possible via atomic operations which are required to be on WB space to work properly. On non-WB spaces, the A/D bit updates are done via bus locks which are not supported for GT. **"1"**: Ignore the page level cacheability and do atomic updates for A/D bit management **"0"**: Detect the page level cacheability as part of the atomic operation and throw a catastrophic error when non-WB space is seen for A/D bit updates.

Replacement

TLB replacements during runtime are based on LRA algorithm; in addition invalidations and page responses will have to invalidate the TLB entries.

Invalidations of TLB

There are various ways to invalidate TLBs:

1. **Traditional invalidation from command streamer**: Could be part of any fence accesses including newly added atomics
2. **SVM based invalidations**: Listed as part of the new SVM related invalidations, various stages of TLBs including intermediate stages can be invalidated selectively and/or as a whole.
3. **Context Switch**: A context switch has to invalidate caches to make sure we have no residual value of the TLBs across multiple PASIDs. GAM will treat the context reload message from CS as a form of TLB invalidation.
4. **A page response**: should invalidate faulted recordings. It should be done via address matching to kick the faulted entries within the matching PASID.

Invalidation response "Invalidation Wait Descriptor" should also be a fence for both READs and WRITEs that used the previous TLB entries. GAM can only respond to "invalidation wait descriptor" after getting a GTI EMPTY indication.

Optional Invalidations

The following cases are listed as page table updates which software may choose not to invalidate the TLBs.

- If a paging-structure is modified to change the Present (Valid) flag from 0 to 1, s/w may choose not to invalidate TLBs. This affects only the case where GPU keeps the faulted page in its TLB to filter out future faults. Regardless of s/w does invalidation or not, for the cases where h/w cares, there will be a page response from s/w which will be used to shootdown the faulted record from the TLB.

GAM will put faulted entries to its TLBs only if there has been page request for it, which means that only faultable surfaces can be stored in GAM TLBs as a faulted entry.

- If a paging-structure entry is modified to change the accessed flag from 0 to 1,no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W or U/S or XD flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address

GTT Walk Request Port (HDC)

A private GTT request port has been added between the HDC(s) and GAM to service the page walks. HDC clusters will contain a mini-TLB and uses GAM's page walker. Their accesses to this page walker is provided thru this private ports. Main GAM TLBs also act as a secondary cache to back these TLBs.

When page walk request comes to GAM, it will be treated as any normal request where the TLB look up will be done and in case of a miss further page walk will be performed. The results of the page walk will be returned on the private connection between the GAM and HDC clusters.

The hierarchy is defined as following diagram where each slice will contain a "Slice GTT Request Manager" (slice GRM) where all HDCs interface with. Each HDC get two credits (i.e. 2-deep ingress queue per HDC) where walk request response back to HDC is considered the release of credits. Slice GRM will collect the walk requests and arbitrate/forward them to GAM on per slice dedicated port.

The request interface is designed to support 1 page walk request per 4 core clocks. Hence both the HDC to slice GRM and slice GRM to GAM should be designed to carry a single page request distributed over 4 clocks to keep the wiring at minimum.

Page Request Interface:

- Valid – 1 bit
- Opcode – 1 bit ("0": Page Request and "1": TLB Invalidation Response)
- Slice ID – 1 bit
- HDC ID – 2 bits
- Virtual (GFX) Address – 36 bits (corresponds to [47:12])
- R/W – Read vs Write intend – 1 bit
- Tracking Number – 8 bits
- Faultable vs non-fautable surface - 1 bits

Page response interface from GAM is designed to deliver one page response every 4 clocks and it is broadcast bus that connects to all HDCs directly. It is up to HDC unit to decode slice/unit ID and claim the response as its own which is also treated as claiming the page miss credit back.

Page Response Interface:

- Valid – 1 bit
- Opcode – 1 bit
  - 00: for Page Response
  - 01: Reserved
  - 10: TLB Invalidation Start
  - 11: TLB Invalidation End
- PA – 27 bits (corresponds to [38:12])
- R/W – this was for a read or write
- Slice ID – 2 bit
- HDC ID – 2 bits
- D bit – 1 bit
- Fault Codes – 2 bits (6 bits)
- Cacheability (memory type) Override – 3 bits
- Tracking number – 8 bits

Fault Codes:

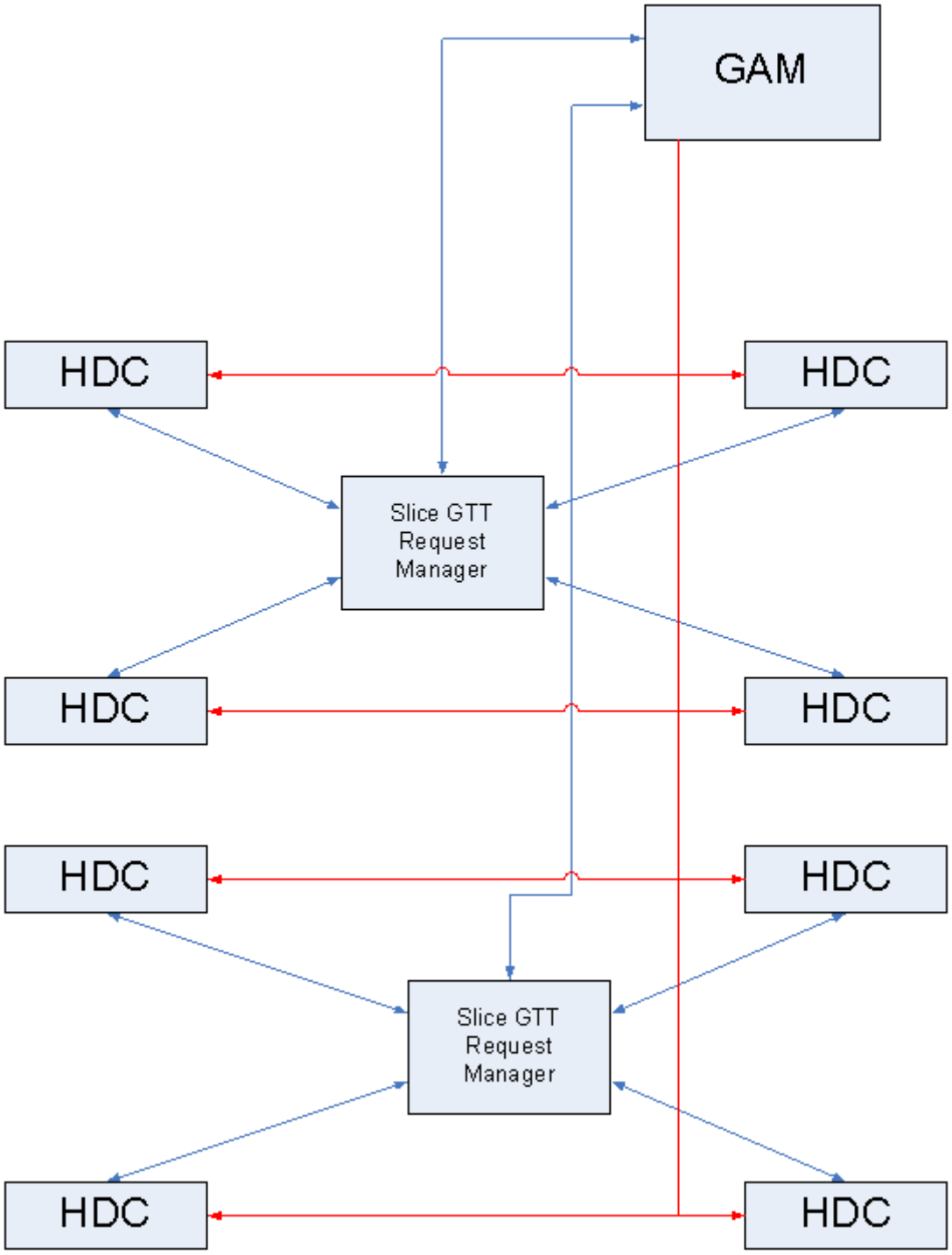
Bits[1:0]	Bit Description
00	No Faults
01	Page fault due to Page not present
10	Privilege level violation
11	Write permission violation

Cacheability (memory type) Override – In case of advanced context execution (where HDC coherent mode is only applicable), the cacheability from surface state will need to be overridden by the OS/VMM setting up the page tables (PAT), MTRR and CD. The effective memory type for HDC has to be used for cache allocation starting with L3. HDC needs to use the memory type bits reported by GAM for memory accesses.

Memory Type	Encoding in MTRR	HDC to L3 Control[3:2]
Uncacheable (UC)	0h	"00"
Write Combining (WC)	1h	"01"
Write Through (WT)	4h	"10"
Write Protected (WP) (Reads:WB and Writes:WC)	5h	Read: "11" Write: "01"
WriteBack (WB)	6h	"11"
Reserved*	2,3,7h	Reserved

*HDC is already capable of processing WT and WB memory types

Overall Signaling Diagram for HDC/GAM connection:



TLB Invalidation

In addition to page walk requests, there is also a communication needed between HDCs and GAM to relate the TLB invalidation events. GAM combines all TLB invalidation events into a single event as a global TLB invalidation to HDC where the entire content of mini-TLB is wiped out.

The protocol starts with GAM sending a “TLB invalidation start” on *page response* interface. All HDCs will act on the TLB invalidation request as it is a broadcast event. Inline communication of the TLB invalidation is to make sure all previous page responses are seen by the HDCs targeted. Upon receiving the TLB Invalidation start, HDCs will stop sending new TLB requests and only process already available translations pending and ensure corresponding (physical accesses) are GO’ed by L3. Once all these steps are complete HDC will send out the ACK on the “page request” interface to GAM.

GAM will stop sending any page responses post “TLB invalidation start” message and it is free to drop any new request that might have been enqueued by HDCs prior to HDCs seeing the invalidation request. The inline ACK from each HDC is meant to push pending HDC TLB requests towards GAM (where they are dropped). Once GAM collects all “TLB invalidations ACK’s” from all HDCs, it will re-enable the TLB service path and send back (broadcast) “TLB invalidation end” message (inline).

HDCs seeing the “TLB Invalidation end” indicating the sequence are complete and synchronized are free to send TLB requests back to GAM.

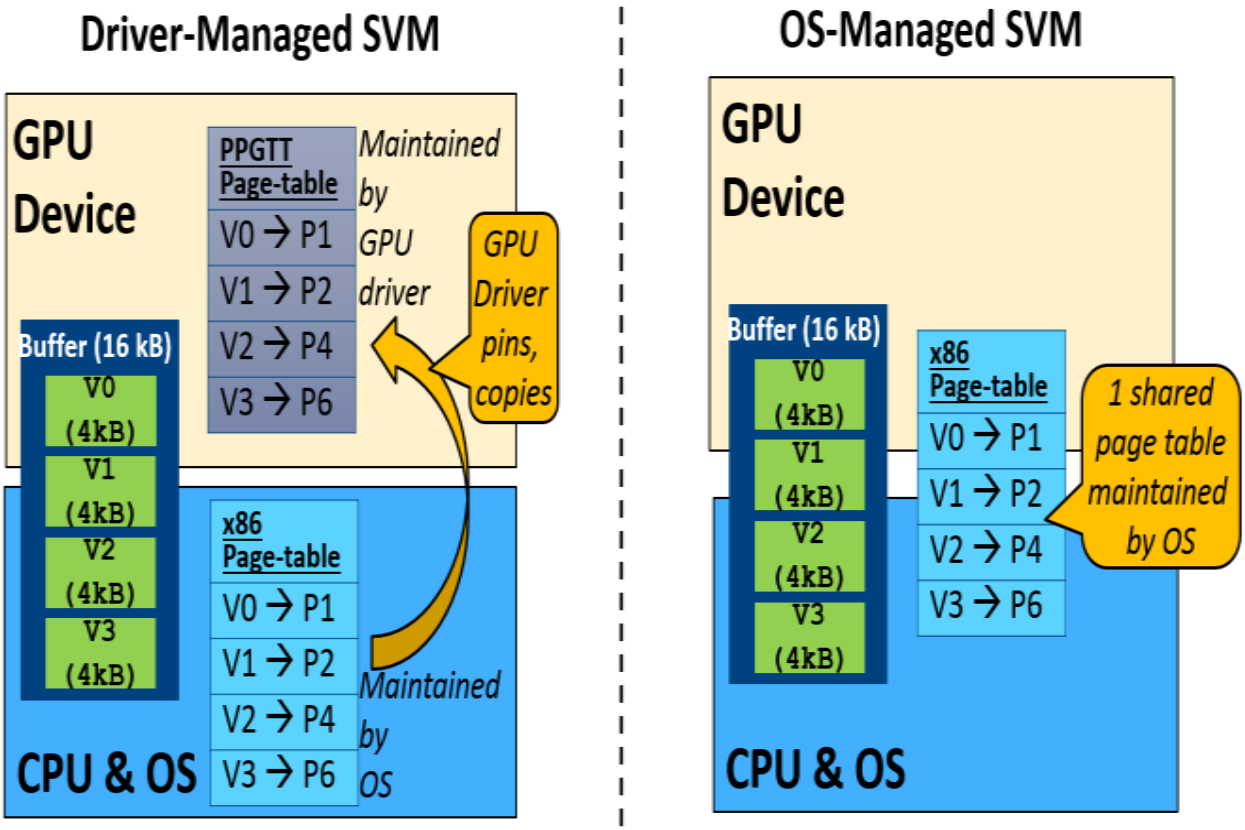
Shared Virtual Memory

The GPU can share virtual memory with the CPU. This is accomplished with Page Tables that map each shared virtual address for both CPU application and the GPU context to the same physical address. There are 2 ways to manage the page tables for shared virtual memory: Driver-Managed SVM and OS-Managed SVM.

Shared Virtual Memory Summary

	Driver-Managed SVM 2 page tables: PPGTT and OS PT	OS-Managed SVM 1 shared page table: OS PT
Page Table type	PPGTT	VT-d IOMMU
Tiled Resource Table	Yes (Driver-Managed TR-TT)	Yes (Driver-Managed TR-TT)
GPU Context Type	Legacy	Advanced
Page Fault Resiliency	None (Fault-and-Hang)	Yes (Fault-and-Halt, Fault-and-Stream)
Coherency	Both Driver-Managed and HW-Managed	Both Driver-Managed and HW-Managed

Page Table Types



Driver-Managed SVM uses 2 different page tables which the graphics driver keeps synchronized: the PPGTT and the OS page table. The driver copies OS page table entries into the GPU context's local PPGTT page table, and pins those physical pages in the OS page table so that OS doesn't change the mapping independent of the driver.

OS-Managed SVM uses 1 shared page table that only the OS manages. The OS coordinates the GPU and CPU use of the shared page table with the VT-d IOMMU. (See IOMMU section for more information on this.)

Page Faults

OS page management services may change a physical memory page mapping that is in use by the GPU, thereby exposing GPU virtual address translations to page faults. The GPU implements a "Fault-and-Halt" resiliency scheme to resolve such page faults: the GPU signals the page fault back to the OS, the OS corrects the memory mapping, and then the OS signals the GPU to retry the memory operation.

When the GPU is running in Advanced Context mode with OS-Managed SVM, this Fault-and-Halt page fault mode is enabled.

When the GPU is running in Legacy Context mode with Driver-Managed SVM, an invalid page translation causes the context to hang ("Fault-and-Hang" resiliency scheme). Driver-Managed SVM strictly avoids the Fault-and-Hang situation: the driver changes the page mappings in PPGTT when the GPU context is not active, and maps all invalid pages to null pages.

Programming Note	
Context:	Fault-and-Stream support
In early configurations, there were two page fault modes supported: Fault-and-Halt and Fault-and-Stream. Most GPU programs use Fault-and-Halt, but GPU programs using coherent memory must use Fault-and-Stream mode. In later configurations, only Fault-and-Halt page fault mode is used.	

SVM Coherency

Memory accesses made by the CPU and GPU are not necessarily coherent (consistent). Each device may have cached copies of the memory data, and require special protocols to keep them coherent. Coarse-grained SVM coherency uses a driver-managed programming model for consistency. Fine-grained SVM coherency uses a HW-managed protocol (MESI) for consistency.

	SW Coarse-Grain Coherency	HW Fine-Grain Coherency
Driver Managed (Mirrored Page Tables via PPGTT)	**OpenCL "Coarse Grain SVM (buffer)"** Programmer benefits: • Pointer based data structure sharing between CPU & GPU	**OpenCL "Fine Grain SVM (buffer)"** Programmer benefits: • Pointer based data structure sharing between CPU & GPU • Concurrent & coherent CPU & GPU memory accesses • Synchronization via CPU & GPU shared memory atomic operations
OS Managed (Shared Page Tables via VTd-IOMMU	This combo does not exist in OpenCL specification	**OpenCL "Fine Grain SVM System"** Programmer benefits: Same as Fine Grain SVM, plus: • Use C/C++ malloc(), new() for memory allocation sharing b/t CPU & GPU • GPU can over-allocate physical memory. (Potential/Eventual) Platform benefits : • OS can centralize memory management • GPU may avoid overcommitting system memory or needlessly taxing system • Simpler driver code (no pinning)

Page Faults

Page fault handling has been fundamentally updated for production processor graphics. Previous generations of production GPUs had limited page faulting capabilities enabled, where such occurrences are either handled as hardware hangs or just simply ignored.

This generation introduces the Advanced Context mode and the Fault&Stream page fault mode as the preferred page fault model into production GPUs. The Advanced Context enables a context switch while a fault handler is run and then a switch back to resume the program. This enables shared virtual memory (SVM) support for GPGPU applications for dynamic allocation of the pages. There is limited support of the Legacy Fault&Halt page fault mode (which halts the Gen engine while the page fault is handled) to facilitate bringing up the new driver independent of bringing up the new page fault model.

This generation moves the page fault mode selection from context descriptor to per-surface state definition via memory object control state (MOCS). This capability enables the GFX driver to define a unique page fault mode at the surface level. H/W allows multiple different types of page fault modes to co-exists. The usage mode for per-surface fault context is exclusively either:

- The compatible Context Register page fault mode with every per-surface MOCS set to “use context register”, or
- The per-surface model where the Context Register page fault mode is set to Fault&Halt and the per-surface page fault mode is any of the defined modes.

Page Walker (GAM) is the gateway to page tables, hence it is responsible to detect and handle the first level requirements for page faulting. The rest of the section is introduced to explain high level treatment of page faults in the hardware.

Hardware and Software Pieces for Page Faults

Various levels of software and hardware entities are involved in GPGPU/3D workloads and interact as part of page fault handling.

Entity	Description
KMD (Kernel Mode Driver)	Responsible for configuring/initialization of device, owns Graphics contexts/work-queue, performs privileged operations for GPGPU contexts (such as setup/teardown of application queue/ring-buffer, etc.).
UMD (User Mode Driver) & App	Capable of submitting commands directly from ring-3 for GPGPU applications/processes via the ring-buffer/queue allocated to it by the KMD. (The current expectation is that ring-buffer is allocated by the user mode driver and pinned by the kernel mode driver.)
IOMMU SW (and/or OS)	For IOMMU enabled future OSs, this would be the native OS kernel. For legacy OSs (i.e. not IOMMU aware), this would be a kernel-mode IOMMU driver (which may be packaged as part of the GPU/GPU software drivers/binaries). For early generations of page fault handling a companion driver model can be used for IOMMU SW. This mode is called SVM_DEVICE_MODE to indicate that, if enabled by BIOS, the IOMMU hardware is accessible to KMD through the Device-2 register resources, so that KMD can manage the IOMMU hardware for legacy (SVM-IOMMU-unaware) OSs.
Memory Interface Controller (Page Walker/GAM)	Hardware component responsible for doing the page walks, identifying page faults, and managing TLBs. The same interface manages the fault queues to/from SW, and reports out faults to other HW entities.
EUs (Execution Units)	Source of the memory accesses via messages, in this context handles the page fault response. Saves its context when we switch out, or retries the faulted transactions when told to do so. Also detects canonical errors.

Page Access Rights

The accesses permitted by a translation are determined by the access rights specified by the paging-structure entries controlling the translation. For legacy context, the access rights are not applicable and should not be considered during page walk.

The following items detail how paging determines access rights during Advanced Contexts:

- For accesses in supervisor context mode:
  - Data reads (Read requests with value of 0 in Execute-Requested (ER) field)
    - Data reads are allowed from any input address with a valid translation.
  - Instruction Fetches (Read requests with value of 1 in Execute-Requested (ER) field)
    - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 0
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 0, instruction may be fetched from any input address with a valid translation.
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 1, instruction may be fetched from any input address with a valid translation for which the U/S field (bit 2) is 0 in at least one of the paging-structure entries controlling the translation.
    - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 1
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 0, instruction may be fetched from any input address with a valid translation for which the XD field (bit 63) is 0 in every paging-structure entry controlling the translation
      - If Supervisor-Mode-Execute-Protection (SMEP) field in extended-context-entry used to translate request is 1, instruction may be fetched from any input address with a valid translation for which, the U/S field is 0 in at least one of the paging-structure entries controlling the translation, and the XD field is 0 in every paging-structure entry controlling the translation.
  - Write requests and Atomics requests
    - If Write-Protect-Enable (WPE) field in extended-context-entry used to translate request is 0, writes are allowed to any input address with a valid translation.
    - If WPE=1, writes are allowed to any input address with a valid translation for which the R/W field (bit 1) is 1 in every paging-structure entry controlling the translation.
    -
- For accesses in user context mode:
  - Data reads (Read requests with value of 0 in Execute-Requested (ER) field)

        - Data reads are allowed from any input address with a valid translation for which the U/S field is 1 in every paging-structure entry controlling the translation.
    - Instruction fetches (Read requests with value of 1 in Execute-Requested (ER) field)
        - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 0, instructions may be fetched from any input address with a valid translation for which the U/S field is 1 in every paging structure entry controlling the translation.
        - If No-Execute-Enable (NXE) field in extended-context-entry used to translate request is 1, instructions may be fetched from any input address with a valid translation for which the U/S field is 1 and XD field is 0 in every paging-structure entry controlling the translation.
    - Write requests and Atomics requests
        - Writes are allowed to any input address with a valid translation for which the R/W field and the U/S field are 1 in every paging-structure entry controlling the translation.

A GPU may cache information from the paging-structure entries in TLBs and paging-structure caches. These structures may include information about access rights. The GPU may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

Page Fault Decision

Only PPGTT/IA-32e page-table accesses are faultable, GGTT accesses are pinned pages and ensured by KMD. This policy is not enforced in the hardware.

All surfaces are page faultable with the introduction into production of Fault&Halt mode.

Fault Aware Surfaces

The concept is that only advanced contexts can be stream page-faultable for clients that are fault aware, and handling mechanism for faults is defined as part of the context itself. GAM will get the fault handling policy as part of the context descriptor ahead of any memory accesses. Another point is that only PPGTT/IA-32e page-table accesses are faultable, GGTT accesses are pinned pages and ensured by KMD, this policy is not enforced in the hardware.

Both front pipeline accesses and backend pipeline accesses are not advanced faultable.

It is easier to limit the surfaces that are faultable than not faultable:

- HDC – Data port accesses (coherent only)
- IC – Instruction cache accesses
- Sampler – Sampler cache accesses

Any other accesses are handled as fault&halt if it is chosen to be so.

GAM will figure out the fault&stream capability of the surface based on the TAG coming from L3 (original requester) and/or based on the GTT paging request from HDC.

Page Fault Modes

Arbiters and memory subsystem will have different faulting modes and behaviors. The following fault behaviors are decided either as part of the context descriptor and/or surface state (MOCS).

To avoid the replay buffer limitations when using Fault & Stream, sampler accesses are handled as Fault&Halt. Advanced 3D and GPGPU contexts still need to be programmed as running under Fault & Stream mode to enable invalidations of TLB translations for coherent surfaces. Potential deadlocks can occur if Fault & Stream is not enabled for data port (HDC) accesses, as coherent cycles may not be able to get flushed out in the case of a TLB invalidation. Programming the context as Fault & Stream enables the replay buffering in the EUs, which is needed to avoid the deadlocks due to TLB invalidations of coherent surfaces. However, the actual fault handling mode in GAM is to be programmed as Fault & Halt, for all faults from all clients. This is done by setting 0x404C[0] = '1.

Context Type		Supported Fault Types	Context Switching on Faults	Notes
Legacy 3d		No Page Fault	n/a	
Legacy Media		No Page Fault	n/a	Programmable Media pipeline
Legacy GPGPU		No Page Fault	n/a	
Advanced 3d		No Page Fault	n/a	
	data port accesses	F&H	No context switch on F&H	
	instruction cache	F&H	No context switch on F&H	
	Sampler cache accesses	F&H	No context switch on F&H	
	all other surfaces	F&H	No context switch on F&H	
Advanced Media		F&H	No context switch on F&H	GPGPU Programmable Media pipeline must be F&H mode for IME/VME Sampler accesses
Advanced GPGPU		F&H or F&S	Context switch on F&S page fault	
	data port accesses	F&H or F&S	Context switch on F&S page fault	Coherent L3 accesses require F&S mode, and will context switch on F&S. Non-coherent accesses can run in either mode but will not context switch if faulted.
	instruction cache	F&H or F&S	Context switch on F&S page fault	

	Sampler cache accesses	F&H or F&S	No context switch on F&S	Sampler accesses are running under F&H mode
	all other surfaces	F&H or F&S	No context switch on F&S	All other accesses are running under F&H mode

Page Faultable Context?	No	Yes	Yes
Fault Mode	Non-fault	Fault and Halt	Fault & Stream
Context Type	Legacy or Advanced	Advanced	Advanced
Shared Func. Support	n/a	All	Sampler, HDC (coherent), IC
Fault Counters Inc'd	No	No	Yes
IOMMU Action On Fault	Signal fatal error	Page Fill	Page Fill
Shared Func Action on Fault	<oblivious>	Wait for fill	Continue; may retry subsequent msgs to ensure ordering rules
Msg Retry (2)	n/a	No	Yes
Kernel Visibility of Fault	None	None	None
App Visibility of Fault	Via driver signaling of Fatal Error	None	None
Scheduler Visibility of Fault	n/a	None	Yes; IRQ on first fault & counters
Usage	Legacy Behavior		

Fault and Hang/Crash (Legacy Mode)

GPU cores prior to this release all implemented Fault and Hang behavior (optionally continue – MMIO based) with the exception of a simple fault and halt that was introduced earlier. This is where page walker detects a page which is not present as part of the translation and hangs the pipeline via reporting into a register/interrupt (through command streamer).

The resulting issue points to a s/w problem either in defining the GTT or using a surface which is not meant to be page faultable.

The same behavior will be carried forward and used for legacy context operation as well as optionally for surfaces that are not page fault-able. The same MMIO based mechanism will allow the engine to ignore the page fault and resume operating.

Fault and Halt

Fault&Halt mode is a simple way of handling page faults where GPU stalls on the page fault and expects services from page miss handler. The context that is faulted cannot be switched out until fault services are complete.

Fault & Halt has a single mechanism to work with:

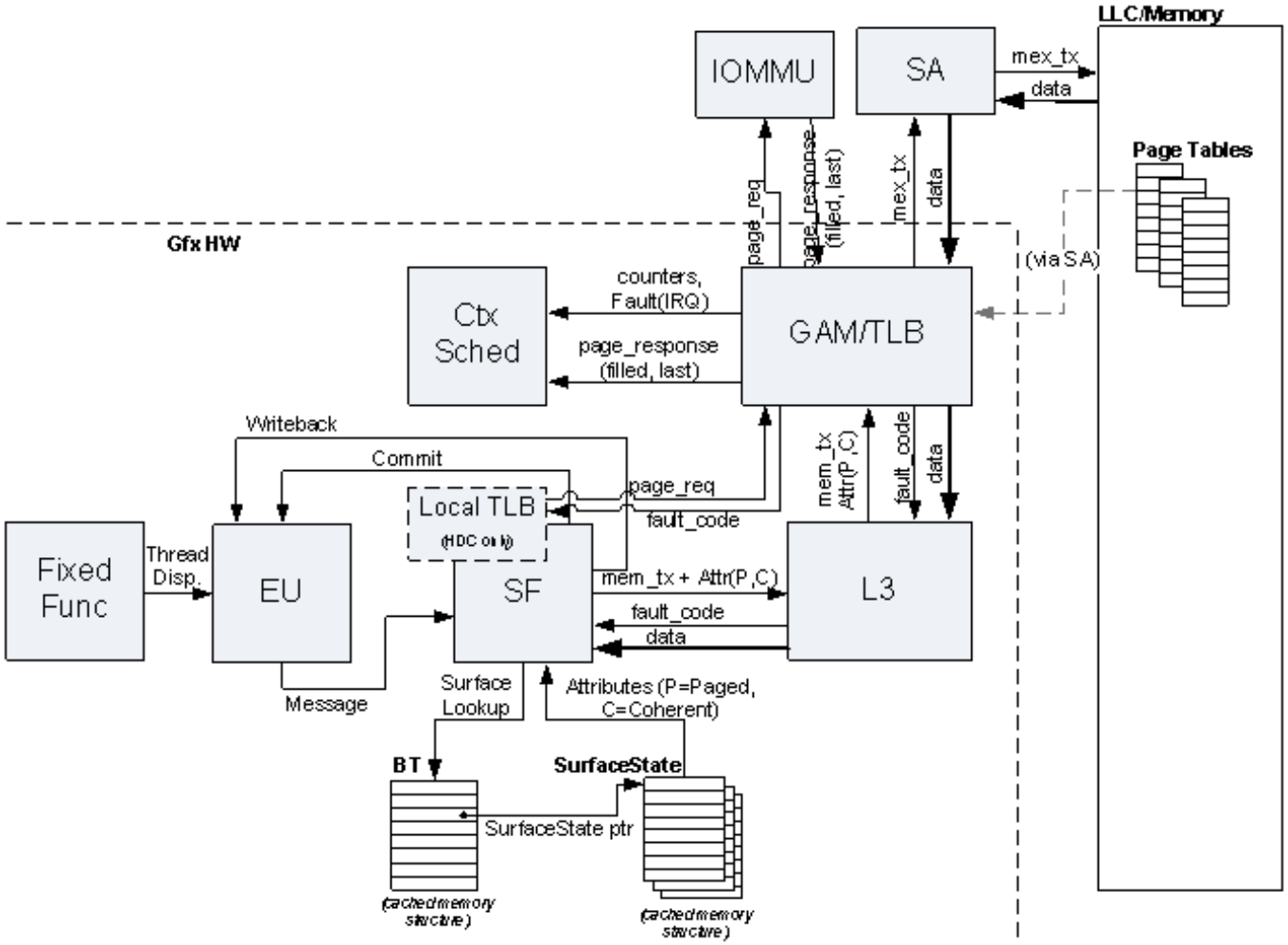
1. Advanced Reporting: As part of the IOMMU additions, page request queue can also be used to report page faults for any levels of paging process. In case of advanced reporting in for Fault & Halt, request queue is used to send the paging request thru IOMMU. This mechanism is primarily used for non-stream fault-able surfaces.

The basic promise of fault & halt is to provide a basic faulting mechanism for non-stream fault-able surfaces regardless of the context type.

As part of the operation, the miss pipeline is blocked until the TOP request is serviced by software. Meanwhile the TLB hits continue to make progress until the TLB MISS-Q fills up blocking the entire memory pipeline.

Fault Flows and Impact

The following diagram shows the relevant streams and fault interfaces for their communication. GAM plays a crucial role in facilitating the page fault information.



On a faultable context GAM is the interfacing HW block for Page Request and Page Response queues. Both queues are used to communicate with SW layer. For some fault modes (see "page fault modes"), the page faults need to be reported to different places. This section outlines these interfaces (excluding the page request/response interface which is tracked in a different section).

Client/GAM Interface

There are two ways that page faultable requests make their way to GAM:

1. Via HDC TLB Request Path: For both writes and reads originated from EUs
2. Shared Functions (Texture & Instructions) Read & Writes: Routed thru L3

HDC GTT request/response interface is already defined as part of the previous sections and the fault code is communicated as part of the page walk response.

There is a need to add new fields to Read Data return interface for communicating the fault information. This information is only relevant for some selective clients (L3, I$, and Sampler) and the rest of the clients do not need to connect the new signals.

Legacy Memory Data Return Interface

Data Valid
Data Tag
Slice ID
Client ID
Data

The legacy return interface is only applicable for Reads, but for faults we need a write response interface not only for reads but also for writes. And the fact that GAM can walk reads and writes separately, we would need an additional response interface for writes (for faultable write clients – HDC only).

Memory Data Return Interface

Data Valid
Data Tag
Slice ID
Client ID
Data
Read Page Fault Code

The read return interface is managed by GTI on a regular scheduling basis. For GAM to intervene and insert a page fault response a "gap" is needed on the read return bus from GTI. However there is no guarantee a "gap" would be present on streaming returns which will lead to infinite holding of page fault responses in GAM. Note page faults do not generate a read to memory hence no data phase (data content is X for the faulted read return). To solve this issue a new signal needs to be inserted from GAM to GTI, which a pulse on that signal guarantees a bubble clock on the main read return bus. The bubble is generated within the 16 clocks within the pulse first seen. GAM can put a 16-deep read page fault response queue to insert page fault responses to the client. Note that all fields on the return path should be valid, except data.

A specific protocol to insert bubbles on the data stream from GTI to GAM is not needed given that GAM converts the faulted access to invalid access which would return data from stolen memory. The packet is later intercepted by GAM to insert the fault indicators.

Page Fault Codes Used

Bits[1:0]	Bit Description
00b	No Page Fault
01b	Page not present or reserved bit not 0
10b	Privilege level violation
11b	Write permission violation

GAM fault code output is only connected to L3 banks via GAFS; no other client takes in the fault codes. L3 behavior does not change for the cases where the returning data has no faults. But if there is a fault, L3 should not be filling the line into the cache (treat it as uncacheable) and instead just return the packed back to the client (Sampler, I$) with the fault code.

Rescheduling a Faulted Context

A faultable context, which got pre-empted out before completing for any reason, needs to be scheduled back eventually for completion. If the context fault behavior is "*context switch on faults*", then regardless of the context switch reason, it only makes sense to reschedule this context when the page fault handler returns the completion (response to LAST page fault). This is where memory interface HW informs the scheduler stating that page fault response has been fulfilled.

The scheduler then promotes the context (i.e. queue) to available pool (from pending pool) and makes it available for hardware submission.

The particular context that got switched out could have been "*stream on faults context*"; this is where the context is tolerant to page faults and it has enough parallelism to make limited forward progress with faults. If "*stream on fault context*" switches out, the simple option is to handle same as "*context switch on fault*" type context, where HW adds a last flag and scheduler puts it into pending pool. It will only be available for hardware submission when all related page faults are serviced. In this case HW will ignore the intermediate page responses for the switched out "*stream on fault context*" unless it is a response to LAST fault. This is the POR mode of rescheduling a faulted context.

However this behavior may not be the most optimized way of rescheduling a "*stream on fault context*" type. Given the context is tolerant to faults, as soon as priority context is complete (or preempted out), "*stream on fault context*" can be scheduled. Even though this process is possible in principle, the hardware determination to reschedule a "*stream on fault context*" could be quite complex.

The first difficulty is whether to put a LAST or not when the "*stream on fault context*" gets preempted out. If there is a LAST, that makes the termination cases very hard to determine whether any faults for this "*stream on fault context*" is pending before ensuring the page fault queue is clear. If there is no "LAST" and termination comes in, that means no response from IOMMU, again the same exposure on how to ensure faults are cleared from the system.

There are additional cases where the reason for the preemption out of a "*stream on fault context*" could be the number of page faults we are running into and scheduler statistics kicking in. Once we preempt out, those statistics are cleared and HW cannot track them while running a different context. So the

question comes in when to reschedule the "*stream on fault context*" when it was switched out due to high number of page faults.

Application Termination

There are various reasons why an application can be terminated, a few of them could be:

1. User killed the App
2. OS killed the App
3. H/W detected an unrecoverable error reported to s/w which killed the App.

Such termination cases will be overlapped to same sequence of events to cleanly switch out from the terminated application while ensure all h/w and s/w entities are cleaned up before the same PASIDs can be re-assigned.

Step#1: OS Handling the application termination

OS starts the process to kill the application; it will inform the KMD on application termination. KMD will get the chance to clean up any application state in the device h/w. OS or IOMMU software marks this PASID as "pending termination", causing IOMMU page-fault handler to return page-responses with unsuccessful status for page-requests from this PASID.

Step#2: KMD informs the scheduler (with PASID/QueueID)

KMD is requesting the scheduler to clear the h/w status for the corresponding PASID/QueueID.

Step#3: Scheduler informs the h/w and prepares the h/w to switch out.

Step#4: This is where the h/w is now informed about the App termination. There are various cases for the status of the context:

1. This is the first time h/w is hearing "bad context" and context is active
2. This is the first time h/w is hearing "bad context", however context was switched out already (i.e. preempted out)
3. H/W already detected this context as "bad" (i.e. unrecoverable error).

In all cases h/w will perform switch out from the bad context, and the only remaining piece is the page faults which might have been already reported to IOMMU. Here there are two cases of interest:

1. *There were page faults for this context*: If there were page faults, h/w will put a LAST into page faulting queue and put the context into Pending list for scheduler. Scheduler cannot complete the termination of the context, until h/w sees the response for the LAST from IOMMU or OS. Meanwhile IOMMU or OS can return unsuccessful completion for the LAST but the purpose is here is to ensure the entire page fault queue is cleared up for this context.
2. *There were no page faults for this context*: Than h/w communicates a successful switch out back to scheduler (which means context does not need to be held back in pending queue).

Step#5: Scheduler frees up the queues and PASID

Step#6: Scheduler informs the KMD and KMD cleans up it structures

KMD can de-allocate all kernel mode data structures for this app (i.e. context space,…) and makes the PASID available

Step#7: KMD informs the OS/IOMMU

IOMMU stops "dummy" page responses for this PASID, it removes all the pages assigned to this App. Performs and IOTLB invalidation (could be a redundant step), enables the PASID for future use. After completion of PASID termination, OS/IOMMU software to free this PASID entry and mark it as not-present, along with needed PTE-cache/IOTLB invalidation

Memory Types and Cache Interface

This section has additional information on the types of memory which are accessible via the various GT mechanisms. It includes discussion on how the various paging models are used and accessed. See the Graphics Translation Tables for more detailed discussions on paging models.

This section also includes descriptions of how different surface types (MOCS) can be cached in the L3 and the different behaviors which can be enabled.

Memory Object Control State (MOCS)

The memory object control state defines the behavior of memory accesses beyond the graphics core, graphics data types that allow selective flushing of data from outer caches, and controlling cacheability in the outer caches.

This control uses several mechanisms. Control state for all memory accesses can be defined page by page in the GTT entries. Memory objects that are defined by state per surface generally have additional memory object control state in the state structure that defines the other surface attributes. Memory objects without state defining them have memory object state control defined per class in the STATE_BASE_ADDRESS command, with class divisions the same as the base addresses. Finally, some memory objects only have the GTT entry mechanism for defining this control. The table below enumerates the memory objects and the location of the control state for each:

Memory Object	Location of Control State
surfaces defined by SURFACE_STATE: sampling engine surfaces, render targets, media surfaces, pull constant buffers, streamed vertex buffers	SURFACE_STATE
depth, stencil, and hierarchical depth buffers	corresponding state command that defined the buffer attributes
stateless buffers accessed by data port	STATE_BASE_ADDRESS
indirect state objects	STATE_BASE_ADDRESS
kernel instructions	STATE_BASE_ADDRESS
push constant buffers	3DSTATE_CONSTANT_(VS | GS | PS)
index buffers	3DSTATE_INDEX_BUFFER
vertex buffers	3DSTATE_VERTEX_BUFFERS
indirect media object	STATE_BASE_ADDRESS

Memory Object	Location of Control State
generic state prefetch	GTT control only
ring/batch buffers	GTT control only
context save buffers	GTT control only
store DWord	GTT control only

MOCS Registers

These registers provide the detailed format of the MOCS table entries that need to be programmed to define each surface state.

MEMORY_OBJECT_CONTROL_STATE

L3 Control Registers

64x16b control registers are defined within L3 space to interpret MOCS indexing and map it to cache events.

The incoming MOCS value is used to index into one of these registers which hardware uses as control parameters for a given surface. It allows 64 concurrent surface definitions with unique control values for L3 caching.

Also attached are the default settings for each 64 locations if driver chooses to use as is.

Following 16b defines per selection definition:

Register#64 (MOCS value 63) is reserved for h/w use and should not be used by s/w.

In L3 Node: B020-B09F (128 Bytes)

Bits	Description
16:6	Reserved.
5:4	**L3 Cacheability Control (L3CC).** Memory type information used in L3. This field is combined with the additional two bits that are sent by HDC based on binding table index. For all other L3 requesters, this field is the primary source of L3 cache controls. 00b: Use binding table index for direct EU accesses – for rest it is reserved. 01b: Uncacheable (UC) – non-cacheable. 10b: Reserved 11b: Writeback (WB).

Bits	Description
3:1	**Skip Caching Control (SCC).** Defines the bit values to enable caching. Outcome overrides the L3caching for the surface. If "0" – than corresponding address bit value is don't care. Bit[1]=1: Address bit[9] needs to be "0" to cache in target. Bit[2]=1: Address bit[10] needs to be "0" to cache in target. Bit[3]=1: Address bit[11] needs to be "0" to cache in target.
0	**Enable Skip Caching (ESC).** Enable for the Skip cache mechanism. 0: Not enabled. 1: Enabled for L3.

Defaults Table

Default	L3CC	SCC	ESC		Default	L3CC	SCC	ESC
000000	00	000	0		100000	00	000	0
000001	01	000	0		100001	01	000	0
000010	01	001	1		100010	01	001	1
000011	01	011	1		100011	01	011	1
000100	01	111	1		100100	01	111	1
000101	10	000	0		100101	10	000	0
000110	01	001	1		100110	01	001	1
000111	01	011	1		100111	01	011	1
001000	01	111	1		101000	01	111	1
001001	11	000	0		101001	11	000	0
001010	01	001	1		101010	01	001	1
001011	01	011	1		101011	01	011	1
001100	01	111	1		101100	01	111	1
001101	00	000	0		101101	00	000	0
001110	00	000	0		101110	00	000	0
001111	00	000	0		101111	00	000	0
010000	00	000	0		110000	00	000	0
010001	01	000	0		110001	01	000	0
010010	01	001	1		110010	01	001	1
010011	01	011	1		110011	01	011	1
010100	01	111	1		110100	01	111	1

Default	L3CC	SCC	ESC		Default	L3CC	SCC	ESC
010101	10	000	0		110101	10	000	0
010110	01	001	1		110110	01	001	1
010111	01	011	1		110111	01	011	1
011000	01	111	1		111000	01	111	1
011001	11	000	0		111001	11	000	0
011010	01	001	1		111010	01	001	1
011011	01	011	1		111011	01	011	1
011100	01	111	1		111100	01	111	1
011101	00	000	0		111101	00	000	0
011110	00	000	0		111110	00	000	0
011111	00	000	0		111111	00	000	0

Memory Interface Control Registers

8*64x32b control registers are defined within the page walker where control parameters for LLC/eDRAM caching are defined. Incoming memory control object state index is used to do a look up into the table where the corresponding control parameters are picked for a given surface. These control values are used to control LLC/eDRAM caching.

For EU surfaces where binding table index is used, we also pass two bits of information in the hardware.

Following 32b defines per selection definition:

These set of registers have to be engine specific (8x).

All MOCS registers are considered part of the HW context and need to be saved part of the context that command streamers are controlling.

Register#64/#63 (MOCS value 63&62) are reserved for h/w use and should not be used by s/w.

- Register#64 is for Coherent L3 evictions
- Register#63 is used for Tiled-Resources page walker accesses

In GAMT: C800-CFFF (8x256 Bytes)

Bits	Description
31:19	**Reserved**
18:17	**Reserved**
16:15	**Class of Service** This field controls the Class of Service sent to the LLC to determine which sub-set of Ways the surface will be stored in.The allocation of certain LLC ways to different class of service settings is a project dependent decision and listed in the Bspec.

Bits	Description
	00: Value from Private PAT Registers(40E0/40E4/40E8/40EC) 01: Class 1 10: Class 2 11: Class 3
14	**Snoop Control Field (SCF)**: Enables s/w to have GFX h/w to be able to consume IA generated buffers that are tagged as WB. Driver can mark these buffers as WB when generating them from IA. In LP-SOCs, the fabric is not forced to be coherent all the time. IA-core generated WB buffers can only be consumed by GPU if that buffer is tagged as snoop-able in GPUs buffer definitions (or via GPU Page tables). 1: Hardware will snoop the IA caches while accessing this surface 0: Hardware will not snoop the IA caches while accessing this surface *Note: There is a performance & power penalty in accessing surfaces that are tagged as snooped.* *Note: S/W should NOT set this field in client platforms.*
13:11	**Page Faulting Mode** This fields controls the page faulting mode that will be used in the memory interface block for the given request coming from this surface: 000: Use the global page faulting mode from context descriptor (default) 001-111: Reserved
10:8	**Skip Caching Control** Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" - than corresponding address bit value is do not care Bit[8]=1: address bit[9] needs to be "0" to cache in target Bit[9]=1: address bit[10] needs to be "0" to cache in target Bit[10]=1: address bit[11] needs to be "0" to cache in target The default value of this field is '000. **For coherent surfaces, skip caching should not be enabled, as not caching in LLC breaks the coherency.**
7	**Enable Reverse Skip Caching** Enable for the Skip cache mechanism 0: Not enabled 1: Enabled for LLC

Bits	Description
6	**Don't Allocate on miss** Controls defined for RO surfaces in mind, where if the target cache is missed - do not bring the line (applicable to LLC/eDRAM). 0: Allocate on MISS (normal cache behavior) 1: Do NOT allocate on MISS
5:4	**LRU (Cache Replacement) Management (LRUM).** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age (“3”) it tends to stay longer in the cache as compared to older age allocations (“2”, “1”, or “0”). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 00: Take the age value from Uncore CRs. 01: Assign the age of "0" 10: Dont change the age on a hit. 11: Assign the age of "3"
3:2	**Target Cache (TC).** This field allows the choice of LLC vs eLLC for caching. 00b: Use TC/LRU controls from page table 01b: LLC Only. 10b: LLC/eLLC Allowed. 11b: LLC/eLLC Allowed. **For coherent surfaces ensure that LLC caching is enabled - even when using target cache controls from page table.**
1:0	**LLC/eDRAM Cacheability Control (LeCC).** Memory type information used in LLC/eDRAM. 00b: Use Cacheability Controls from page table / UC with Fence (if coherent cycle). 01b: Uncacheable (UC) – non-cacheable. 10b: Writethrough (WT). 11b: Writeback (WB). **Note:** In case of SVM (advanced context), LLC/eDRAM memory type is used based on the page table controls and cannot be managed via MOCS index.

Defaults Table

Default	LeCC	TC	LRUM	AOM	ESC	SCC	PFM	Default	LeCC	TC	LRUM	AOM	ESC	SCC	PFM
000000	00	00	11	0	0	00	000	000000	00	00	11	0	0	00	000
000001	00	01	11	0	0	00	000	000001	00	01	11	0	0	00	000
000010	00	10	11	0	0	00	000	000010	00	10	11	0	0	00	000
000011	01	00	11	0	0	00	000	000011	01	00	11	0	0	00	000
000100	10	00	11	0	0	00	000	000100	10	00	11	0	0	00	000
000101	10	01	11	0	0	00	000	000101	10	01	11	0	0	00	000
000110	10	10	11	0	0	00	000	000110	10	10	11	0	0	00	000
000111	11	00	11	0	0	00	000	000111	11	00	11	0	0	00	000
001000	11	01	11	0	0	00	000	001000	11	01	11	0	0	00	000
001001	11	10	11	0	0	00	000	001001	11	10	11	0	0	00	000
001010	10	00	11	0	0	00	000	001010	10	00	11	0	0	00	000
001011	10	01	11	0	0	00	000	001011	10	01	11	0	0	00	000
001100	10	10	11	0	0	00	000	001100	10	10	11	0	0	00	000
001101	11	00	11	0	0	00	000	001101	11	00	11	0	0	00	000
001110	11	01	11	0	0	00	000	001110	11	01	11	0	0	00	000
001111	11	10	11	0	0	00	000	001111	11	10	11	0	0	00	000
010000	00	00	11	0	0	00	000	010000	00	00	11	0	0	00	000
010001	00	01	11	0	0	00	000	010001	00	01	11	0	0	00	000
010010	00	10	11	0	0	00	000	010010	00	10	11	0	0	00	000
010011	01	00	11	0	0	00	000	010011	01	00	11	0	0	00	000
010100	10	00	11	0	0	00	000	010100	10	00	11	0	0	00	000
010101	10	01	11	0	0	00	000	010101	10	01	11	0	0	00	000
010110	10	10	11	0	0	00	000	010110	10	10	11	0	0	00	000
010111	11	00	11	0	0	00	000	010111	11	00	11	0	0	00	000
011000	11	01	11	0	0	00	000	011000	11	01	11	0	0	00	000
011001	11	10	11	0	0	00	000	011001	11	10	11	0	0	00	000
011010	10	00	11	0	0	00	000	011010	10	00	11	0	0	00	000
011011	10	01	11	0	0	00	000	011011	10	01	11	0	0	00	000
011100	10	10	11	0	0	00	000	011100	10	10	11	0	0	00	000
011101	11	00	11	0	0	00	000	011101	11	00	11	0	0	00	000
011110	11	01	11	0	0	00	000	011110	11	01	11	0	0	00	000
011111	11	10	11	0	0	00	000	011111	11	10	11	0	0	00	000

HDCL3GAM Change Specific to Coherent L3

Given memory object control state is an index for L3 coherent accesses HDC cannot replace the bit[3:2] with the memory type information coming from GAM. Instead these two bits need to be communicated separately.

Both L3 and GAM override the memory type information that is extracted from the index table with these two bits passed by HDC in case of L3 coherent accesses.

Memory Type	Encoding in MTRR	HDC to L3 Control[3:2]
Uncacheable (UC)	0h	"00"
Write Combining (WC)	1h	"01"
Write Through (WT)	4h	"10"
Write Protected (WP) (Reads:WB and Writes:WC)	5h	Read: "11" Write: "01"
WriteBack (WB)	6h	"11"
Reserved*	2,3,7h	Reserved

Also for the HDC GAM interface (page request response), the faulting mode from indexed table needs to be communicated along with response. This would add 2 bits and HDC can use the mode bits to figure out what to do with the data.

This is also applicable to read return data from GAM.

Graphics Cache and Memory Interface

The memory interface has further improvements over previous generations. These improvements are either on existing functions or new features that are added to this particular generation.

Skip Caching in LLC and eDRAM

Skip Caching is added to deal with two specific problems:

1. If a given surface is too large for LLC or eDRAM, we want to cache a portion of the surface in the cache and get limited benefits instead of trying to cache the entire surface and trash the cache. Or not cache at all and lose any benefit of using a cache.
2. If a given surface can benefit from additional/concurrent b/w of both caches and/or memory, we can split the surface via allocating a certain segments of it in different caches and memory, and try to extract additional b/w. The opportunity would be additional b/w from different sources rather being stuck with cache b/w only.

For coherent surfaces, skip caching should not be enabled, as not caching in LLC breaks the coherency.

Mechanism can be controlled via memory object control state which is used to identify each surface separately.

Bit	Description
10:8	**Skip Caching Control (SCC).** Defines the bit values to enable caching. Outcome overrides the LLC caching for the surface. If "0" then corresponding address bit value is don't care. • Bit[8]=1: Address bit[9] needs to be "0" to cache in target. • Bit[9]=1: Address bit[10] needs to be "0" to cache in target. • Bit[10]=1: Address bit[11] needs to be "0" to cache in target.
7	**Enable Reverse Skip Caching (ESC).** Enable for the Skip cache mechanism. • 0: Not enabled. • 1: Enabled for reverse caching.

Here is the mechanism:

If bit[7]=0:

- If access is LLC cacheable and skip caching is enabled, match the enabled address bits:
  - If matched address bits, let the caching to be in LLC
  - If no-matched address bits, downgrade the caching to eLLC
- If access is eLLC/eDRAM cacheable (Only) and skip caching is enabled, match the enabled address bits:

- If matched address bits, let the caching to be in eLLC
- If no-matched address bits, downgrade the memory type to UC (uncacheable).

If bit[7]=1:

- If access is LLC cacheable and skip caching is enabled, match the enabled address bits:
  - If matched address bits, downgrade the caching to eLLC
  - If no-matched address bits, let the caching to be in LLC
- If access is eLLC/eDRAM cacheable (Only) and skip caching is enabled, match the enabled address bits:
  - If matched address bits, downgrade the memory type to UC (uncacheable).
  - If no-matched address bits, let the caching to be in eLLC

Caching Display Surfaces in LLC

Using LLC for the output Color pipeline when Display buffers are accessed has been tried for many generations. Previous generations enabled write-through caching to get benefits from reads. Generation GFX adds a mechanism to be able to cache display buffers that are immediate target of the color pipeline as write-back. In order to synchronize the contents of the display buffer within LLC to display controller, PG added an option at the pipe-control flush to select whether a color buffer flush is needed along with pipeline flush (see pipe-control and MI_Flush_DW definition updates).

For Frame buffer caching GFX driver will tag the corresponding Display buffer target as cacheable in LLC. LLC cacheability shall be selected via GFX Page tables (using private PAT) or MOCS (memory object control state). However this setting would enable for display buffers to be cached along with general GFX accesses which could lead to trashing in LLC cache along with long flush penalties. In order to limit the implications of an LLC flush, a global setting register (*IDICR – IDI Control Register[23:22]*) is included to control the QOS (quality of service) value for display buffers only. GFX driver is required to tune the QOS value based on system settings to limit the number of ways required to be allocated for frame buffer caching and to be flushed at the FLIP time.

For proper LLC flush event, GFX driver will program the cache definition registers (*FBCDR – Frame Buffer Cache Definition Register*) in the Page Walker register space to define different parameters of the LLCcache on the system.

Pipe-control and/or MI_FLUSH_DW for the contexts that have Frame Buffer caching has to be enhanced to enable overlap execution of the LLC flush as well as the execution of next context/frame by h/w. Not doing the following steps will cause GT h/w to stall on LLC flush which is not desirable.

In the case where Frame Buffer is cached, driver is required to put 2 pipe-controls and/or MI_FLUSH_DW's.

- The first will be used to flush the GPU h/w with a stall, and will not have LLC flush or SYNC'ing DW/interrupt to the driver.
- Second will be used to flush LLC with a SYNC DW/interrupt to the driver. This will not be a stalling pipe-control and/or MI_FLUSH_DW.

After first pipe-control/MI_FLUSH_DW, hardware will be free to execute forward on the command ring and 2nd pipe-control/MI_FLUS_DW will be non-stalling for h/w but will hold off the FLIP synchronization to display controller until display buffer contents are flushed from LLC.

Page Walker Access and Memory Types

Most of these notes are further explained in the document however summarized as part of the page table behavior:

Page Walker Memory Types

1. Legacy Contexts
    a. GT access to root/extended-root table and context/extended-context table
    b. GTT access to private paging (PPGTT) entries
    c. GT access to GPA-to-HPA paging entries
    d. GT access to the translated page
2. Advanced context (without nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
3. Advanced context (with nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
    e. GT access to GPA-to-HPA paging entries to translate address of PASID-entry and PASID-state entry
    f. GT access to GPA-to-HPA paging entries to translate address of IA-32e paging entries
    g. GT access to GPA-to-HPA paging entries to translate address of page

Error Cases

- A/D bit update attempt for paging entry in non-WB memory, cause page-walk to be aborted; Error reported to device in Translation Response; For Gen, gets reported to driver as GPGPU context in error – catastrophic error case.
- Locked/Atomic operations to pages in non-WB memory aborted; For Gen, gets reported to driver as GPGPU context in error (catastrophic error)
- CD=1 treated same as non-WB memory, for above lock behavior

Common Surface Formats

This section documents surfaces and how they are stored in memory, including 3D and video surfaces, including the details of compressed texture formats. Also covered are the surface layouts based on tiling mode and surface type.

Non-Video Surface Formats

This section describes the lowest-level organization of a surfaces containing discrete “pixel” oriented data (e.g., discrete pixel (RGB,YUV) colors, subsampled video data, 3D depth/stencil buffer pixel formats, bump map values etc. Many of these pixel formats are common to the various pixel-oriented memory object types.

Surface Format Naming

Unless indicated otherwise, all pixels are **stored** in “**little endian**” byte order. i.e., pixel bits 7:0 are stored in byte *n*, pixel bits 15:8 are stored in byte *n*+1, and so on. The format labels include color components in little endian order (e.g., R8G8B8A8 format is physically stored as R, G, B, A).

The name of most of the surface formats specifies its format. Channels are listed in little endian order (LSB channel on the left, MSB channel on the right), with the channel format specified following the channels with that format. For example, R5G5_SNORM_B6_UNORM contains, from LSB to MSB, 5 bits of red in SNORM format, 5 bits of green in SNORM format, and 6 bits of blue in UNORM format.

Intensity Formats

All surface formats containing “I” include an intensity value. When used as a source surface for the sampling engine, the intensity value is replicated to all four channels (R,G,B,A) before being filtered. Intensity surfaces are not supported as destinations.

Luminance Formats

All surface formats containing “L” include a luminance value. When used as a source surface for the sampling engine, the luminance value is replicated to the three color channels (R,G,B) before being filtered. The alpha channel is provided either from another field or receives a default value. Luminance surfaces are not supported as destinations.

R1_UNORM (same as R1_UINT) and MONO8

When used as a texel format, the R1_UNORM format contains 8 1-bit Intensity (I) values that are replicated to all color channels. Note that T0 of byte 0 of a R1_UNORM-formatted texture corresponds to Texel[0,0]. This is different from the format used for monochrome sources in the BLT engine.

7	6	5	4	3	2	1	0
T7	T6	T5	T4	T3	T2	T1	T0

Bit	Description
T0	**Texel 0** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1
...	**...**
T7	**Texel 7** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1

MONO8 format is identical to R1_UNORM but has different semantics for filtering. MONO8 is the only supported format for the MAPFILTER_MONO filter. See the *Sampling Engine* chapter.

Palette Formats

Palette formats are supported by the sampling engine. These formats include an index into the palette (Px) that selects the actual channel values from the palette, which is loaded via the 3DSTATE_SAMPLER_PALETTE_LOAD0 command.

P4A4_UNORM

This surface format contains a 4-bit Alpha value (in the high nibble) and a 4-bit Palette Index value (in the low nibble).

7			4	3			0
Alpha				Palette Index			

Bit	Description
7:4	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] Alpha value. Format: U4
3:0	**Palette Index** A 4-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U4

A4P4_UNORM

This surface format contains a 4-bit Alpha value (in the low nibble) and a 4-bit Color Index value (in the high nibble).

7			4	3			0
Palette Index				Alpha			

Bit	Description
7:4	**Palette Index** A 4-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U4

Bit	Description
3:0	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] alpha value. Format: U4

P8A8_UNORM

This surface format contains an 8-bit Alpha value (in the high byte) and an 8-bit Palette Index value (in the low byte).

15			8	7			0
Alpha				Palette Index			

Bit	Description
15:8	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] Alpha value. Format: U8
7:0	**Palette Index** An 8-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U8

A8P8_UNORM

This surface format contains an 8-bit Alpha value (in the low byte) and an 8-bit Color Index value (in the high byte).

15			8	7			0
Palette Index				Alpha			

Bit	Description
15:8	**Palette Index** An 8-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U8

Bit	Description
7:0	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] alpha value. Format: U8

P8_UNORM

This surface format contains only an 8-bit Color Index value.

Bit	Description
7:0	**Palette Index** An 8-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U8

P2_UNORM

This surface format contains only a 2-bit Color Index value.

Bit	Description
1:0	**Palette Index** A 2-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U2

Compressed Surface Formats

This section contains information on the internal organization of compressed surface formats.

ETC1_RGB8

This format compresses UNORM RGB data using an 8-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows.

High 24 bits if "diff" is zero (individual mode):

Bits	Description
7:4	R0[3:0]
3:0	R1[3:0]
15:12	G0[3:0]
11:8	G1[3:0]
23:20	B0[3:0]
19:16	B1[3:0]

High 24 bits if "diff" is one (differential mode):

Bits	Description
7:3	R0[4:0]
2:0	dR1[2:0]
15:11	G0[4:0]
10:8	dG1[2:0]
23:19	B0[4:0]
18:16	dB1[2:0]

Low 40 bits:

Bits	Description
31:29	lum table index for sub-block 0
28:26	lum table index for sub-block 1
25	diff
24	flip
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB
36	texel[0][3] index MSB
35	texel[3][2] index MSB

Bits	Description
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB
40	texel[0][0] index MSB
55	texel[3][3] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The 4x4 is divided into two 8-pixel sub-blocks, either two 2x4 sub-blocks or two 4x2 sub-blocks controlled by the “flip” bit. If flip=0, sub-block 0 is the 2x4 on the left and sub-block 1 is the 2x4 on the right. If flip=1, sub-block 0 is the 4x2 on the top and sub-block 1 is the 4x2 on the bottom.

The “diff” bit controls whether the red/green/blue values (R0/G0/B0/R1/G1/B1) are stored as one 444 value per sub-block (“individual” mode with diff = 0), or a single 555 value for the first sub-block (R0/G0/B0) and a 333 delta value (dR1/dG1/dB1) for the second sub-block (“differential” mode with diff = 1). The delta values are 3-bit two’s-complement values that hold values in the range [-4,3]. These values are added to the 5-bit values for sub-block 0 to obtain the 5-bit values for sub-block 1 (if the value is outside of the range [0,31], the result of the decompression is undefined). From the 4- or 5-bit

per channel values, an 8-bit value for each channel is extended by replication and provides the 888 base color for each sub-block.

For each sub-block one of 8 different luminance columns is selected based on the 3-bit lum table index. Then each texel selects one of the 4 rows of the selected column with a 2-bit per-texel index. The chosen value in the table is added to the 8-bit base color for the sub-block (obtained in the previous step) to obtain the texel's color. Values in the table are given in decimal, representing an 8-bit UNORM as an 8-bit signed integer.

Luminance Table

	0	1	2	3	4	5	6	7
0	2	5	9	13	18	24	33	47
1	8	17	29	42	60	80	106	183
2	-2	-5	-9	-13	-18	-24	-33	-47
3	-8	-17	-29	-42	-60	-80	-106	-183

ETC2_RGB8 and ETC2_SRGB8

The ETC2_RGB8 format builds on top of ETC1_RGB8, using a set of invalid bit sequences to enable three new modes. The two modes of ETC1_RGB8 are also supported with ETC2_RGB8, and will not be documented in this section as they are covered in the ETC1_RGB8 section.

The detection of the three new modes is based on RGB and diff bits in locations as defined for ETC1 differential mode. The mode is determined as follows (x indicates don't care):

diff	Rt	Gt	Bt	mode
0	x	x	x	individual
1	0	x	x	T
1	1	0	x	H
1	1	1	0	planar
1	1	1	1	differential

The inputs in the above table are defined as follows:

```
Rt = (R0 + dR1) in [0,31]
Gt = (G0 + dG1) in [0,31]
Bt = (G0 + dB1) in [0,31]
```

8-byte compression block for mode determination

Bits	Description
7:3	R0[4:0]
2:0	dR1[2:0]
15:11	G0[4:0]
10:8	dG1[2:0]
23:19	B0[4:0]
18:16	dB1[2:0]
31:26	ignored
25	diff
24	ignored
63:32	ignored

The fields in the table above are used *only* for mode determination. Some of the bits in this table are overloaded with other values within each mode. The algorithm is defined such that there is no ambiguity in modes when this is done.

T mode

The "T" mode has the following bit definition:

8-byte compression block for "T" mode

Bits	Description
7:5	ignored
4:3	R0[3:2]
2	ignored
1:0	R0[1:0]
15:12	G0[3:0]
11:8	B0[3:0]
23:20	R1[3:0]
19:16	G1[3:0]
31:28	B1[3:0]
27:26	di[2:1]
25	diff = 1
24	di[0]
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB

Bits	Description
36	texel[0][3] index MSB
35	texel[3][2] index MSB
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB
40	texel[0][0] index MSB
55	texel[0][0] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The “T” mode has two base colors stored as 4 bits per channel, R0/G0/B0 and R1/G1/B1, as in the individual mode, however the bit positions for these are different. For each channel, the 4 bits are extended to 8 bits by bit replication.

A 3-bit distance index “di” is also defined in the compression block. This value is used to look up the distance in the following table:

distance index "di"	distance "d"
0	3
1	6
2	11
3	16
4	23
5	32
6	41
7	64

Four colors are possible on each texel. These colors are defined as the following:

```
P0 = (R0, G0, B0)
P1 = (R1, G1, B1) + (d, d, d)
P2 = (R1, G1, B1)
P3 = (R1, G1, B1) - (d, d, d)
```

All resulting channels are clamped to the range [0,255]. One of the four colors is then assigned to each texel in the block based on the 2-bit texel index.

H mode

The "H" mode has the following bit definition:

8-byte compression block for "H" mode

Bits	Description
7	ignored
6:3	R0[3:0]
2:0	G0[3:1]
15:13	ignored
12	G0[0]
11	B0[3]
10	ignored
9:8	B0[2:1]
23	B0[0]
22:19	R1[3:0]
18:16	G1[3:1]
31	G1[0]
30:27	B1[3:0]
26	di[2]

Bits	Description
25	diff = 1
24	di[1]
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB
36	texel[0][3] index MSB
35	texel[3][2] index MSB
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB
40	texel[0][0] index MSB
55	texel[3][3] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The “H” mode has two base colors stored as 4 bits per channel, R0/G0/B0 and R1/G1/B1, as in the individual and T modes, however the bit positions for these are different. For each channel, the 4 bits are extended to 8 bits by bit replication.

A 3-bit distance index “di” is defined by 2 MSBs in the compression block and the LSB computed by the following equation, where R/G/B values are the 8-bit values from the first step:

```
di[0] = ((R0 « 16) | (G0 « 8) | B0) >= ((R1 « 16) | (G1 « 8) | B1)
```

The distance “d” is then looked up in the same table used for T mode. The four colors for H mode are computed as follows:

```
P0 = (R0, G0, B0) + (d, d, d)
P1 = (R0, G0, B0) - (d, d, d)
P2 = (R1, G1, B1) + (d, d, d)
P3 = (R1, G1, B1) - (d, d, d)
```

All resulting channels are clamped to the range [0,255]. One of the four colors is then assigned to each texel in the block based on the 2-bit texel index as in T mode.

Planar mode

The “planar” mode has the following bit definition:

8-byte compression block for “planar” mode

Bits	Description
7	ignored
6:1	R0[5:0]
0	G0[6]
15	ignored
14:9	G0[5:0]
8	B[5]
23:21	ignored
20:19	B[4:3]
18	ignored
17:16	B0[2:1]
31	B0[0]
30:26	RH[5:1]
25	diff = 1
24	RH[0]
39:33	GH[6:0]
32	BH[5]
47:43	BH[4:0]

Bits	Description
42:40	RV[5:3]
55:53	RV[2:0]
52:48	GV[6:2]
63:62	GV[1:0]
61:56	BV[5:0]

The “planar” mode has three base colors stored as RGB 676, with red & blue having 6 bits and green having 7 bits. These three base colors are each extended to RGB 888 with bit replication.

The color of each texel is then computed using the following equations, with x and y representing the texel position within the compression block:

```
texel[y][x].R = x(RH-R0)/4 + y(RV-R0)/4 + R0
texel[y][x].G = x(GH-G0)/4 + y(GV-G0)/4 + G0
texel[y][x].B = x(BH-B0)/4 + y(BV-B0)/4 + B0
```

All resulting channels are clamped to the range [0,255].

The ETC2_SRGB8 format is decompressed as if it is ETC2_RGB8, then a conversion from the resulting RGB values to SRGB space is performed.

EAC_R11 and EAC_SIGNED_R11

These formats compress UNORM/SNORM single-channel data using an 8-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows.

EAC_R11 compression block layout

Bits	Description
7:0	R0[7:0]
15:12	m[3:0]
11:8	ti[3:0]
23:21	texel[0][0] index
20:18	texel[1][0] index
17:16,31	texel[2][0] index
30:28	texel[3][0] index
27:25	texel[0][1] index
24,39:38	texel[1][1] index
37:35	texel[2][1] index
34:32	texel[3][1] index
47:45	texel[0][2] index

Bits	Description
44:42	texel[1][2] index
41:40,55	texel[2][2] index
54:52	texel[3][2] index
51:49	texel[0][3] index
48,63:62	texel[1][3] index
61:59	texel[2][3] index
58:56	texel[3][3] index

The “ti” (table index) value from the compression block is used to select one of the columns in the table below.

Intensity modifier (im) table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-3	-3	-2	-2	-3	-3	-4	-3	-2	-2	-2	-2	-3	-1	-4	-3
1	-6	-7	-5	-4	-6	-7	-7	-5	-6	-5	-4	-5	-4	-2	-6	-5
2	-9	-10	-8	-6	-8	-9	-8	-8	-8	-8	-8	-7	-7	-3	-8	-7
3	-15	-13	-13	-13	-12	-11	-11	-11	-10	-10	-10	-10	-10	-10	-9	-9
4	2	2	1	1	2	2	3	2	1	1	1	1	2	0	3	2
5	5	6	4	3	5	6	6	4	5	4	3	4	3	1	5	4
6	8	9	7	5	7	8	7	7	7	7	7	6	6	2	7	6
7	14	12	12	12	11	10	10	10	9	9	9	9	9	9	8	8

The eight possible color values R_i are then computed from the 8 values in the column labeled im_i, where i ranges from 0 to 7:

For EAC_R11:

```
if (m == 0) Ri = R0*8 + 4 + imi else Ri = R0*8 + 4 + (imi * m * 8)
```

Each value is clamped to the range [0,2047].

For EAC_SIGNED_R11:

```
if (m == 0) Ri = R0*8 + imi else Ri = R0*8 + (imi * m * 8)
```

Each value is clamped to the range [-1023,1023].

Note that in the signed case, the R0 value is a signed, 2's complement value in the range [-127, 127]. Before being used in the above equations, an R0 value of -128 must be clamped to -127.

Finally, each texel red value is selected from the 8 possible values R_i using the 3-bit index for that texel. The green, blue, and alpha values are set to their default values.

The final value represents an 11-bit UNORM or SNORM as an unsigned/signed integer.

ETC2_RGB8_PTA and ETC2_SRGB8_PTA

The ETC2_RGB8_PTA format is similar to ETC2_RGB8 but eliminates the "individual" mode in favor of allowing a punch-through alpha. The "diff" bit from ETC2_RGB8 is renamed to "opaque" in this format, and the mode selection behaves as if the "diff" bit is always 1, making the "individual" mode inaccessible for these formats.

An alpha value of either 0 or 255 (representing 0.0 or 1.0) is possible with this format. If alpha is determined to be zero, the three other channels are also forced to zero, regardless of what value the normal decompression algorithm would have produced.

Differential Mode

In differential mode, if the opaque bit is set, the luminance table for ETC2_RGB8 is used. If the opaque bit is not set, the following luminance table is used (note that rows 0 and 2 have been zeroed out, otherwise the table is the same):

Luminance Table for opaque bit not set

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	8	17	29	42	60	80	106	183
2	0	0	0	0	0	0	0	0
3	-8	-17	-29	-42	-60	-80	-106	-183

For each texel, if the opaque bit is zero and the corresponding texel index is equal to 2, the alpha value is set to zero (and therefore RGB for that texel will also end up at zero). Otherwise alpha is set to 255 and RGB is the result of the normal decompression calculations.

T and H Modes

In both of these modes, if the opaque bit is zero and the texel index is equal to 2, the alpha value is set to zero (and therefore RGB will also end up at zero). Otherwise alpha is set to 255.

Planar Mode

In planar mode, the opaque bit is ignored and alpha is set to 255.

The ETC2_SRGB8_PTA format is decompressed as if it is ETC2_RGB8_PTA, then a conversion from the resulting RGB values to SRGB space is performed, with alpha remaining unchanged.

ETC2_EAC_RGBA8 and ETC2_EAC_SRGB8_A8

The ETC2_EAC_RGBA8 format is a combination of ETC2_RGB8 and EAC_R8. A 16-byte compression block represents each 4x4. The low-order 8 bytes are used to compute alpha (instead of red) using the EAC_R8 algorithm. The high-order 8 bytes are used to compute RGB using the ETC2_RGB8 algorithm. The EAC_R8 format differs from EAC_R11 as described below.

The ETC2_EAC_SRGB8_A8 format is decompressed as if it is ETC2_EAC_RGBA8, then a conversion from the resulting RGB values to SRGB space is performed, with alpha remaining unchanged.

EAC_R8 Format:

The EAC_R8 format used within these surface formats is identical to EAC_R11 described in an earlier section, except the procedure for computing the eight possible color values Ri is performed as follows:

Ri = R0 + (imi * m)

Each value is clamped to the range [0,255].

EAC_RG11 and EAC_SIGNED_RG11

These formats compress UNORM/SNORM double-channel data using a 16-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 16-byte compression block is laid out as follows.

EAC_RG11 compression block layout

Bits	Description
63:56	G0[7:0]
55:52	Gm[3:0]
51:48	Gti[3:0]
47:45	texel[0][0] G index
44:42	texel[1][0] G index
41:39	texel[2][0] G index
38:36	texel[3][0] G index
35:33	texel[0][1] G index
32:30	texel[1][1] G index
29:27	texel[2][1] G index
26:24	texel[3][1] G index

Bits	Description
23:21	texel[0][2] G index
20:18	texel[1][2] G index
17:15	texel[2][2] G index
14:12	texel[3][2] G index
11:9	texel[0][3] G index
8:6	texel[1][3] G index
5:3	texel[2][3] G index
66:64	texel[3][3] G index
63:56	R0[7:0]
55:52	Rm[3:0]
51:48	Rti[3:0]
47:45	texel[0][0] R index
44:42	texel[1][0] R index
41:39	texel[2][0] R index
38:36	texel[3][0] R index
35:33	texel[0][1] R index
32:30	texel[1][1] R index
29:27	texel[2][1] R index
26:24	texel[3][1] R index
23:21	texel[0][2] R index
20:18	texel[1][2] R index
17:15	texel[2][2] R index
14:12	texel[3][2] R index
11:9	texel[0][3] R index
8:6	texel[1][3] R index
5:3	texel[2][3] R index
2:0	texel[3][3] R index

These compression formats are identical to the EAC_R11 and EAC_SIGNED_R11 formats, except that they supply two channels of output data, both red and green, from two independent 8-byte portions of the compression block. The low half of the compression block contains the red information, and the high half contains the green information. Blue and alpha channels are set to their default values.

Refer to the EAC_R11 and EAC_SIGNED_R11 specification for details on how the red and green channels are generated using the data in the compression block.

FXT Texture Formats

There are four different FXT1 compressed texture formats. Each of the formats compress two 4x4 texel blocks into 128 bits. In each compression format, the 32 texels in the two 4x4 blocks are arranged according to the following diagram:

FXT1 Encoded Blocks

t0	t1	t2	t3
t4	t5	t6	t7
t8	t9	t10	t11
t12	t13	t14	t15

t16	t17	t18	t19
t20	t21	t22	t23
t24	t25	t26	t27
t28	t29	t30	t31

B6682-01

Overview of FXT1 Formats

During the compression phase, the encoder selects one of the four formats for each block based on which encoding scheme results in best overall visual quality. The following table lists the four different modes and their encodings:

FXT1 Format Summary

Bit 127	Bit 126	Bit 125	Block Compression Mode	Summary Description
0	0	X	**CC_HI**	2 R5G5B5 colors supplied. Single LUT with 7 interpolated color values and transparent black
0	1	0	**CC_CHROMA**	4 R5G5B5 colors used directly as 4-entry LUT.
0	1	1	**CC_ALPHA**	3 A5R5G5B5 colors supplied. LERP bit selects between 1 LUT with 3 discrete colors + transparent black and 2 LUTs using interpolated values of Color 0,1 (t0-15) and Color 1,2 (t16-31).
1	x	x	**CC_MIXED**	4 R5G5B5 colors supplied, where Color0,1 LUT is used for t0-t15, and Color2,3 LUT used for t16-31. Alpha bit selects between LUTs with 4 interpolated colors or 3 interpolated colors + transparent black.

FXT1 CC_HI Format

In the CC_HI encoding format, two base 15-bit R5G5B5 colors (Color 0, Color 1) are included in the encoded block. These base colors are then expanded (using high-order bit replication) to 24-bit RGB colors, and used to define an 8-entry lookup table of interpolated color values (the 8th entry is

transparent black). The encoded block contains a 3-bit index value per texel that is used to lookup a color from the table.

CC_HI Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_HI block format:

FXT CC_HI Block Encoding

Bit	Description
127:126	Mode = ‘00’b (CC_HI)
125:121	Color 1 Red
120:116	Color 1 Green
115:111	Color 1 Blue
110:106	Color 0 Red
105:101	Color 0 Green
100:96	Color 0 Blue
95:93	Texel 31 Select
...	...
50:48	Texel 16 Select
47:45	Texel 15 Select
...	...
2:0	Texel 0 Select

CC_HI Block Decoding

The two base colors, Color 0 and Color 1 are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

FXT CC_HI Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [23:19]	Color 1 Red [7:3]	[125:121]
Color 1 [18:16]	Color 1 Red [2:0]	[125:123]
Color 1 [15:11]	Color 1 Green [7:3]	[120:116]
Color 1 [10:08]	Color 1 Green [2:0]	[120:118]
Color 1 [07:03]	Color 1 Blue [7:3]	[115:111]
Color 1 [02:00]	Color 1 Blue [2:0]	[115:113]
Color 0 [23:19]	Color 0 Red [7:3]	[110:106]
Color 0 [18:16]	Color 0 Red [2:0]	[110:108]
Color 0 [15:11]	Color 0 Green [7:3]	[105:101]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 0 [10:08]	Color 0 Green [2:0]	[105:103]
Color 0 [07:03]	Color 0 Blue [7:3]	[100:96]
Color 0 [02:00]	Color 0 Blue [2:0]	[100:98]

These two 24-bit colors (Color 0, Color 1) are then used to create a table of seven interpolated colors (with Alpha = 0FFh), along with an eight entry equal to RGBA = 0,0,0,0, as shown in the following table:

FXT CC_HI Interpolated Color Table

Interpolated Color	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(5 * Color0.RGB + 1 * Color1.RGB + 3) / 6	0FFh
2	(4 * Color0.RGB + 2 * Color1.RGB + 3) / 6	0FFh
3	(3 * Color0.RGB + 3 * Color1.RGB + 3) / 6	0FFh
4	(2 * Color0.RGB + 4 * Color1.RGB + 3) / 6	0FFh
5	(1 * Color0.RGB + 5 * Color1.RGB + 3) / 6	0FFh
6	Color1.RGB	0FFh
7	RGB = 0,0,0	0

This table is then used as an 8-entry Lookup Table, where each 3-bit Texel n Select field of the encoded CC_HI block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_HI block.

FXT1 CC_CHROMA Format

In the CC_CHROMA encoding format, four 15-bit R5B5G5 colors are included in the encoded block. These colors are then expanded (using high-order bit replication) to form a 4-entry table of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

CC_CHROMA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_CHROMA block format:

FXT CC_CHROMA Block Encoding

Bit	Description
127:125	Mode = '010'b (CC_CHROMA)
124	Unused
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue

Bit	Description
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	
33:32	Texel 16 Select
31:30	Texel 15 Select
...	
1:0	Texel 0 Select

CC_CHROMA Block Decoding

The four colors (Color 0-3) are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

FXT CC_CHROMA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10:08]	Color 3 Green [2:0]	[118:116]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

This table is then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_CHROMA block is used to index into a 32-bit A8R8G8B8 color from the table (Alpha defaults to 0FFh) completing the decode of the CC_CHROMA block.

FXT CC_CHROMA Interpolated Color Table

Texel Select	Color ARGB
0	Color0.ARGB
1	Color1.ARGB
2	Color2.ARGB
3	Color3.ARGB

FXT1 CC_MIXED Format

In the CC_MIXED encoding format, four 15-bit R5G5B5 colors are included in the encoded block: Color 0 and Color 1 are used for Texels 0-15, and Color 2 and Color 3 are used for Texels 16-31.

Each pair of colors are then expanded (using high-order bit replication) to form 4-entry tables of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

CC_MIXED Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_MIXED block format:

FXT CC_MIXED Block Encoding

Bit	Description
127	Mode = ‘1’b (CC_MIXED)
126	Color 3 Green [0]
125	Color 1 Green [0]

Bit	Description
124	Alpha [0]
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

CC_MIXED Block Decoding

The decode of the CC_MIXED block is modified by Bit 124 (Alpha [0]) of the encoded block.

Alpha[0] = 0 Decoding

When Alpha[0] = 0 the four colors are encoded as 16-bit R5G6B5 values, with the Green LSB defined as per the following table:

FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 2 Green [0]	Encoded Bit [33] XOR Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]
Color 0 Green [0]	Encoded Bit [1] XOR Encoded Bit [125]

The four colors (Color 0-3) are then converted from R5G5B6 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

FXT CC_MIXED Decoded Colors (Alpha[0] = 0)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10]	Color 2 Green [2]	[33] XOR [126]]
Color 2 [09:08]	Color 2 Green [1:0]	[103:100]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10]	Color 0 Green [2]	[1] XOR [125]
Color 0 [09:08]	Color 0 Green [1:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four interpolated colors (with Alpha = 0FFh). The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices, as shown in the following figures:

FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh

Texel 0-15 Select	Color RGB	Alpha
1	(2*Color0.RGB + Color1.RGB + 1) /3	0FFh
2	(Color0.RGB + 2*Color1.RGB + 1) /3	0FFh
3	Color1.RGB	0FFh

FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(2/3) * Color2.RGB + (1/3) * Color3.RGB	0FFh
2	(1/3) * Color2.RGB + (2/3) * Color3.RGB	0FFh
3	Color3.RGB	0FFh

Alpha[0] = 1 Decoding

When Alpha[0] = 1, Color0 and Color2 are encoded as 15-bit R5G5B5 values. Color1 and Color3 are encoded as RGB565 colors, with the Green LSB obtained as shown in the following table:

FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]

All four colors are then expanded to 24-bit R8G8B8 colors by bit replication, as show in the following diagram.

FXT CC_MIXED Decoded Colors (Alpha[0] = 1)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:19]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:87]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:19]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices. The color at index 1 is the linear interpolation of the base colors, while the color at index 3 is defined as Black (0,0,0) with Alpha = 0, as shown in the following figures:

FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(Color0.RGB + Color1.RGB) /2	0FFh
2	Color1.RGB	0FFh
3	Black (0,0,0)	0

FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(Color2.RGB + Color3.RGB) /2	0FFh
2	Color3.RGB	0FFh
3	Black (0,0,0)	0

These tables are then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_MIXED block is used to index into the appropriate 32-bit A8R8G8B8 color from the table, completing the decode of the CC_CMIXED block.

FXT1 CC_ALPHA Format

In the CC_ALPHA encoding format, three A5R5G5B5 colors are provided in the encoded block. A control bit (LERP) is used to define the lookup table (or tables) used to dereference the 2-bit Texel Selects.

CC_ALPHA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_ALPHA block format:

FXT CC_ALPHA Block Encoding

Bit	Description
127:125	Mode = '011'b (CC_ALPHA)
124	LERP
123:119	Color 2 Alpha
118:114	Color 1 Alpha
113:109	Color 0 Alpha
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

CC_ALPHA Block Decoding

Each of the three colors (Color 0-2) are converted from A5R5G5B5 to A8R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

FXT CC_ALPHA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 2 [31:27]	Color 2 Alpha [7:3]	[123:119]
Color 2 [26:24]	Color 2 Alpha [2:0]	[123:121]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [31:27]	Color 1 Alpha [7:3]	[118:114]
Color 1 [26:24]	Color 1 Alpha [2:0]	[118:116]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [31:27]	Color 0 Alpha [7:3]	[113:109]
Color 0 [26:24]	Color 0 Alpha [2:0]	[113:111]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

LERP = 0 Decoding

When LERP = 0, a single 4-entry lookup table is formed using the three expanded colors, with the 4th entry defined as transparent black (ARGB=0,0,0,0). Each 2-bit Texel n Select field of the encoded CC_ALPHA block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_ALPHA block.

FXT CC_ALPHA Interpolated Color Table (LERP=0)

Texel Select	Color	Alpha
0	Color0.RGB	Color0.Alpha
1	Color1.RGB	Color1.Alpha
2	Color2.RGB	Color2.Alpha
3	Black (RGB=0,0,0)	0

LERP = 1 Decoding

When LERP = 1, the three expanded colors are used to create two tables of four interpolated colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color1,2 table used for texels 16-31 indices, as shown in the following figures:

FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 0-15)

Texel 0-15 Select	Color ARGB
0	Color0.ARGB
1	(2*Color0.ARGB + Color1.ARGB + 1) /3
2	(Color0.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 16-31)

Texel 16-31 Select	Color ARGB
0	Color2.ARGB
1	(2*Color2.ARGB + Color1.ARGB + 1) /3
2	(Color2.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

DXT/BC1-3 Texture Formats

Note that non-power-of-2 dimensioned maps may require the surface to be padded out to the next multiple of four texels – here the pad texels are not referenced by the device.

An 8-byte (QWord) block encoding can be used if the source texture contains no transparency (is opaque) or if the transparency can be specified by a one-bit alpha. A 16-byte (DQWord) block encoding can be used to support source textures that require more than one-bit alpha: here the 1st QWord is used to encode the texel alpha values, and the 2nd QWord is used to encode the texel color values.

These three types of format are discussed in the following sections:

- Opaque and One-bit Alpha Textures (DXT1)
- Opaque Textures (DXT1_RGB)
- Textures with Alpha Channels (DXT2-5)

DXT2 and DXT3 are equivalent compression formats from the perspective of the hardware. The only difference between the two is the use of pre-multiplied alpha encoding, which does not affect hardware.

Likewise, DXT4 and DXT5 are the same compression formats with the only difference being the use of pre-multiplied alpha encoding.

Note that the surface formats DXT1-5 are referred to in the DirectX Specification as BC1-3. The mapping between formats is shown below:

- DXT1 ⇒ BC1
- DXT2/DXT3 ⇒ BC2
- DXT4/DXT5 ⇒ BC3

Programming Note	
Context:	DXT Texture Formats
• Any single texture must specify that its data is stored as 64 or 128 bits per group of 16 texels. If 64-bit blocks—that is, format DXT1—are used for the texture, it is possible to mix the opaque and one-bit alpha formats on a per-block basis within the same texture. In other words, the comparison of the unsigned integer magnitude of color_0 and color_1 is performed uniquely for each block of 16 texels. • When 128-bit blocks are used, then the alpha channel must be specified in either explicit (format DXT2 or DXT3) or interpolated mode (format DXT4 or DXT5) for the entire texture. Note that as with color, once interpolated mode is selected then either 8 interpolated alphas or 6 interpolated alphas mode can be used on a block-by-block basis. Again the magnitude comparison of alpha_0 and alpha_1 is done uniquely on a block-by-block basis.	

Opaque and One-bit Alpha Textures (DXT1/BC1)

Texture format DXT1 is for textures that are opaque or have a single transparent color. For each opaque or one-bit alpha block, two 16-bit R5G6B5 values and a 4x4 bitmap with 2-bits-per-pixel are stored. This totals 64 bits (1 QWord) for 16 texels, or 4-bits-per-texel.

In the block bitmap, there are two bits per texel to select between the four colors, two of which are stored in the encoded data. The other two colors are derived from these stored colors by linear interpolation.

The one-bit alpha format is distinguished from the opaque format by comparing the two 16-bit color values stored in the block. They are treated as unsigned integers. If the first color is greater than the second, it implies that only opaque texels are defined. This means four colors will be used to represent the texels. In four-color encoding, there are two derived colors and all four colors are equally distributed in RGB color space. This format is analogous to R5G6B5 format. Otherwise, for one-bit alpha transparency, three colors are used and the fourth is reserved to represent transparent texels. Note that the color blocks in DXT2-5 formats strictly use four colors, as the alpha values are obtained from the alpha block .

In three-color encoding, there is one derived color and the fourth two-bit code is reserved to indicate a transparent texel (alpha information). This format is analogous to A1R5G5B5, where the final bit is used for encoding the alpha mask.

The following piece of pseudo-code illustrates the algorithm for deciding whether three- or four-color encoding is selected:

```
if (color_0 > color_1)
{
  // Four-color block: derive the other two colors.
  // 00 = color_0, 01 = color_1, 10 = color_2, 11 = color_3
  // These two bit codes correspond to the 2-bit fields
  // stored in the 64-bit block.  color_2 = (2 * color_0 + color_1) / 3;
   color_3 = (color 0 + 2 * color_1) / 3;
}
else
{
  // Three-color block: derive the other color.  // 00 = color_0, 01 = color_1, 10 = color_2,
  // 11 = transparent.  // These two bit codes correspond to the 2-bit fields
  // stored in the 64-bit block.  color_2 = (color_0 + color_1) / 2;
   color_3 = transparent;
}
```

The following tables show the memory layout for the 8-byte block. It is assumed that the first index corresponds to the y-coordinate and the second corresponds to the x-coordinate. For example, Texel[1][2] refers to the texture map pixel at (x,y) = (2,1).

Here is the memory layout for the 8-byte (64-bit) block:

Word Address	16-bit Word
0	Color_0
1	Color_1
2	Bitmap Word_0
3	Bitmap Word_1

Color_0 and Color_1 (colors at the two extremes) are laid out as follows:

Bits	Color
15:11	Red color component
10:5	Green color component
4:0	Blue color component

Bits	Texel
1:0 (LSB)	Texel[0][0]
3:2	Texel[0][1]
5:4	Texel[0][2]
7:6	Texel[0][3]
9:8	Texel[1][0]
11:10	Texel[1][1]

Bits	Texel
13:12	Texel[1][2]
15:14	Texel[1][3]

Bitmap Word_1 is laid out as follows:

Bits	Texel
1:0 (LSB)	Texel[2][0]
3:2	Texel[2][1]
5:4	Texel[2][2]
7:6	Texel[2][3]
9:8	Texel[3][0]
11:10	Texel[3][1]
13:12	Texel[3][2]
15:14 (MSB)	Texel[3][3]

Example of Opaque Color Encoding

As an example of opaque encoding, we will assume that the colors red and black are at the extremes. We will call red color_0 and black color_1. There will be four interpolated colors that form the uniformly distributed gradient between them. To determine the values for the 4x4 bitmap, the following calculations are used:

```
00 ? color_0
01 ? color_1
10 ? 2/3 color_0 + 1/3 color_1
11 ? 1/3 color_0 + 2/3 color_1
```

Example of One-bit Alpha Encoding

This format is selected when the unsigned 16-bit integer, color_0, is less than the unsigned 16-bit integer, color_1. An example of where this format could be used is leaves on a tree to be shown against a blue sky. Some texels could be marked as transparent while three shades of green are still available for the leaves. Two of these colors fix the extremes, and the third color is an interpolated color.

The bitmap encoding for the colors and the transparency is determined using the following calculations:

```
00 ? color_0
01 ? color_1
10 ? 1/2 color_0 + 1/2 color_1
11 ? Transparent
```

Opaque Textures (DXT1_RGB)

Texture format DXT1_RGB is identical to DXT1, with the exception that the One-bit Alpha encoding is removed. Color 0 and Color 1 are not compared, and the resulting texel color is derived strictly from the Opaque Color Encoding. The alpha channel defaults to 1.0.

Programming Note	
Context:	Opaque Textures (DXT1_RGB)
The behavior of this format is not compliant with the OGL spec.	

Compressed Textures with Alpha Channels (DXT2-5 / BC2-3)

There are two ways to encode texture maps that exhibit more complex transparency. In each case, a block that describes the transparency precedes the 64-bit block already described for DXT1. The transparency is either represented as a 4x4 bitmap with four bits per pixel (explicit encoding), or with fewer bits and linear interpolation analogous to what is used for color encoding.

The transparency block and the color block are laid out as follows:

Word Address	64-bit Block
3:0	Transparency block
7:4	Previously described 64-bit block

Explicit Texture Encoding

For explicit texture encoding (DXT2 and DXT3 formats), the alpha components of the texels that describe transparency are encoded in a 4x4 bitmap with 4 bits per texel. These 4 bits can be achieved through a variety of means such as dithering or by simply using the 4 most significant bits of the alpha data. However they are produced, they are used just as they are, without any form of interpolation.

Note: DirectDraw's compression method uses the 4 most significant bits.

The following tables illustrate how the alpha information is laid out in memory, for each 16-bit word.

This is the layout for Word 0:

Bits	Alpha
3:0 (LSB)	[0][0]
7:4	[0][1]
11:8	[0][2]
15:12 (MSB)	[0][3]

This is the layout for Word 1:

Bits	Alpha
3:0 (LSB)	[1][0]
7:4	[1][1]
11:8	[1][2]
15:12 (MSB)	[1][3]

This is the layout for Word 2:

Bits	Alpha
3:0 (LSB)	[2][0]
7:4	[2][1]
11:8	[2][2]
15:12 (MSB)	[2][3]

This is the layout for Word 3:

Bits	Alpha
3:0 (LSB)	[3][0]
7:4	[3][1]
11:8	[3][2]
15:12 (MSB)	[3][3]

Three-Bit Linear Alpha Interpolation

The encoding of transparency for the DXT4 and DXT5 formats is based on a concept similar to the linear encoding used for color. Two 8-bit alpha values and a 4x4 bitmap with three bits per pixel are stored in the first eight bytes of the block. The representative alpha values are used to interpolate intermediate alpha values. Additional information is available in the way the two alpha values are stored. If alpha_0 is greater than alpha_1, then six intermediate alpha values are created by the interpolation. Otherwise, four intermediate alpha values are interpolated between the specified alpha extremes. The two additional implicit alpha values are 0 (fully transparent) and 255 (fully opaque).

The following pseudo-code illustrates this algorithm:

```
// 8-alpha or 6-alpha block?
if (alpha_0 > alpha_1) {
   // 8-alpha block: derive the other 6 alphas.
   // 000 = alpha_0, 001 = alpha_1, others are interpolated
  alpha_2 = (6 * alpha_0 + alpha_1) / 7;     // Bit code 010
  alpha_3 = (5 * alpha_0 + 2 * alpha_1) / 7; // Bit code 011
  alpha_4 = (4 * alpha_0 + 3 * alpha_1) / 7; // Bit code 100
  alpha_5 = (3 * alpha_0 + 4 * alpha_1) / 7; // Bit code 101
  alpha_6 = (2 * alpha_0 + 5 * alpha_1) / 7; // Bit code 110
  alpha_7 = (alpha_0 + 6 * alpha_1) / 7;     // Bit code 111
 }
else {
   // 6-alpha block: derive the other alphas.
   // 000 = alpha_0, 001 = alpha_1, others are interpolated
  alpha_2 = (4 * alpha_0 + alpha_1) / 5;     // Bit code 010
  alpha_3 = (3 * alpha_0 + 2 * alpha_1) / 5; // Bit code 011
  alpha_4 = (2 * alpha_0 + 3 * alpha_1) / 5; // Bit code 100
  alpha_5 = (alpha_0 + 4 * alpha_1) / 5;     // Bit code 101
  alpha_6 = 0;                               // Bit code 110
  alpha_7 = 255;                             // Bit code 111
}
```

The memory layout of the alpha block is as follows:

Byte	Alpha
0	Alpha_0
1	Alpha_1
2	[0][2] (2 LSBs), [0][1], [0][0]
3	[1][1] (1 LSB), [1][0], [0][3], [0][2] (1 MSB)
4	[1][3], [1][2], [1][1] (2 MSBs)
5	[2][2] (2 LSBs), [2][1], [2][0]
6	[3][1] (1 LSB), [3][0], [2][3], [2][2] (1 MSB)
7	[3][3], [3][2], [3][1] (2 MSBs)

BC4

These formats (BC4_UNORM and BC4_SNORM) compresses single-component UNORM or SNORM data. An 8-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] bit code
21:19	texel[0][1] bit code
24:22	texel[0][2] bit code
27:25	texel[0][3] bit code
30:28	texel[1][0] bit code
33:31	texel[1][1] bit code
36:34	texel[1][2] bit code
39:37	texel[1][3] bit code
42:40	texel[2][0] bit code
45:43	texel[2][1] bit code
48:46	texel[2][2] bit code
51:49	texel[2][3] bit code
54:52	texel[3][0] bit code
57:55	texel[3][1] bit code
60:58	texel[3][2] bit code
63:61	texel[3][3] bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four

interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                              // bit code 000
red_1 = red_1;                              // bit code 001
if (red_0 > red_1) {
    red_2 = (6 * red_0 + 1 * red_1) / 7; // bit code 010
    red_3 = (5 * red_0 + 2 * red_1) / 7; // bit code 011
    red_4 = (4 * red_0 + 3 * red_1) / 7; // bit code 100
    red_5 = (3 * red_0 + 4 * red_1) / 7; // bit code 101
    red_6 = (2 * red_0 + 5 * red_1) / 7; // bit code 110
    red_7 = (1 * red_0 + 6 * red_1) / 7; // bit code 111
}
else {
    red_2 = (4 * red_0 + 1 * red_1) / 5; // bit code 010
    red_3 = (3 * red_0 + 2 * red_1) / 5; // bit code 011
    red_4 = (2 * red_0 + 3 * red_1) / 5; // bit code 100
    red_5 = (1 * red_0 + 4 * red_1) / 5; // bit code 101
    red_6 = UNORM ? 0.0 : -1.0;             // bit code 110 (0 for UNORM, -1 for SNORM)
    red_7 = 1.0;                            // bit code 111
}
```

BC5

These formats (BC5_UNORM and BC5_SNORM) compresses dual-component UNORM or SNORM data. A 16-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 16-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] red bit code
21:19	texel[0][1] red bit code
24:22	texel[0][2] red bit code
27:25	texel[0][3] red bit code
30:28	texel[1][0] red bit code
33:31	texel[1][1] red bit code
36:34	texel[1][2] red bit code
39:37	texel[1][3] red bit code
42:40	texel[2][0] red bit code
45:43	texel[2][1] red bit code
48:46	texel[2][2] red bit code
51:49	texel[2][3] red bit code
54:52	texel[3][0] red bit code
57:55	texel[3][1] red bit code

Bit	Description
60:58	texel[3][2] red bit code
63:61	texel[3][3] red bit code
71:64	green_0
79:72	green_1
82:80	texel[0][0] green bit code
85:83	texel[0][1] green bit code
88:86	texel[0][2] green bit code
91:89	texel[0][3] green bit code
94:92	texel[1][0] green bit code
97:95	texel[1][1] green bit code
100:98	texel[1][2] green bit code
103:101	texel[1][3] green bit code
106:104	texel[2][0] green bit code
109:107	texel[2][1] green bit code
112:110	texel[2][2] green bit code
115:113	texel[2][3] green bit code
118:116	texel[3][0] green bit code
121:119	texel[3][1] green bit code
124:122	texel[3][2] green bit code
127:125	texel[3][3] green bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                              // bit code 000
red_1 = red_1;                              // bit code 001
if (red_0 > red_1) {
    red_2 = (6 * red_0 + 1 * red_1) / 7; // bit code 010
    red_3 = (5 * red_0 + 2 * red_1) / 7; // bit code 011
    red_4 = (4 * red_0 + 3 * red_1) / 7; // bit code 100
    red_5 = (3 * red_0 + 4 * red_1) / 7; // bit code 101
    red_6 = (2 * red_0 + 5 * red_1) / 7; // bit code 110
    red_7 = (1 * red_0 + 6 * red_1) / 7; // bit code 111
}
else {
    red_2 = (4 * red_0 + 1 * red_1) / 5; // bit code 010
    red_3 = (3 * red_0 + 2 * red_1) / 5; // bit code 011
    red_4 = (2 * red_0 + 3 * red_1) / 5; // bit code 100
    red_5 = (1 * red_0 + 4 * red_1) / 5; // bit code 101
    red_6 = UNORM ? 0.0 : -1.0;             // bit code 110 (0 for UNORM, -1 for SNORM)
    red_7 = 1.0;                            // bit code 111
}
```

The same calculations are done for green, using the corresponding reference colors and bit codes.

BC6H

These formats (BC6H_UF16 and BC6H_SF16) compresses 3-channel images with high dynamic range (> 8 bits per channel). BC6H supports floating point denorms but there is no support for INF and NaN, other than with BC6H_SF16 –INF is supported. The alpha channel is not included, thus alpha is returned at its default value.

The BC6H block is 16 bytes and represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel. BC6H has 14 different modes, the mode that the block is in is contained in the least significant bits (either 2 or 5 bits).

The basic scheme consists of interpolating colors along either one or two lines, with per-texel indices indicating which color along the line is chosen for each texel. If a two-line mode is selected, one of 32 partition sets is indicated which selects which of the two lines each texel is assigned to.

Field Definition

There are 14 possible modes for a BC6H block, the format of each is indicated in the 14 tables below. The mode is selected by the unique mode bits specified in each table. The first 10 modes use two lines ("TWO"), and the last 4 use one line ("ONE"). The difference between the various two-line and one-line modes is with the precision of the first endpoint and the number of bits used to store delta values for the remaining endpoints. Two modes (9 and 10) specify each endpoint as an original value rather than using the deltas (these are indicated as having no delta values).

The endpoints values and deltas are indicated in the tables using a two-letter name. The first letter is "r", "g", or "b" indicating the color channel. The second letter is "w", "x", "y", or "z" indicating which of the four endpoints. The first line has endpoints "w" and "x", with "w" being the endpoint that is fully specified (i.e. not as a delta). The second line has endpoints "y" and "z". Modes using ONE mode do not have endpoints "y" and "z" as they have only one line.

In addition to the mode and endpoint data, TWO blocks contain a 5-bit "partition" which selects one of the partition sets, and a 46-bit set of indices. ONE blocks contain a 63-bit set of indices. These are described in more detail below.

Mode 0: (TWO) Red, Green, Blue: 10-bit endpoint, 5-bit deltas

Bit	Description
1:0	mode = 00
2	gy[4]
3	by[4]
4	bz[4]
14:5	rw[9:0]
24:15	gw[9:0]

Bit	Description
34:25	bw[9:0]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 1: (TWO) Red, Green, Blue: 7-bit endpoint, 6-bit deltas

Bit	Description
1:0	mode = 01
2	gy[5]
3	gz[4]
4	gz[5]
11:5	rw[6:0]
12	bz[0]
13	bz[1]
14	by[4]
21:15	gw[6:0]
22	by[5]
23	bz[2]
24	gy[4]
31:25	bw[6:0]
32	bz[3]
33	bz[5]
34	bz[4]
40:35	rx[5:0]

Bit	Description
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]
60:55	bx[5:0]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 2: (TWO) Red: 11-bit endpoint, 5-bit deltas

Green, Blue: 11-bit endpoint, 4-bit deltas

Bit	Description
4:0	mode = 00010
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
39:35	rx[4:0]
40	rw[10]
44:41	gy[3:0]
48:45	gx[3:0]
49	gw[10]
50	bz[0]
54:51	gz[3:0]
58:55	bx[3:0]
59	bw[10]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 3: (TWO) Red, Blue: 11-bit endpoint, 4-bit deltas

Green: 11-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 00110
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[10]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	gw[10]
54:51	gz[3:0]
58:55	bx[3:0]
59	bw[10]
60	bz[1]
64:61	by[3:0]
68:65	ry[3:0]
69	bz[0]
70	bz[2]
74:71	rz[3:0]
75	gy[4]
76	bz[3]
81:77	partition
127:82	indices

Mode 4: (TWO) Red, Green: 11-bit endpoint, 4-bit deltas

Blue: 11-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 01010
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[10]
40	by[4]
44:41	gy[3:0]
48:45	gx[3:0]

Bit	Description
49	gw[10]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bw[10]
64:61	by[3:0]
68:65	ry[3:0]
69	bz[1]
70	bz[2]
74:71	rz[3:0]
75	bz[4]
76	bz[3]
81:77	partition
127:82	indices

Mode 5: (TWO) Red, Green, Blue: 9-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 01110
13:5	rw[8:0]
14	by[4]
23:15	gw[8:0]
24	gy[4]
33:25	bw[8:0]
34	bz[4]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]

Bit	Description
76	bz[3]
81:77	partition
127:82	indices

Mode 6: (TWO) Red: 8-bit endpoint, 6-bit deltas

Green, Blue: 8-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 10010
12:5	rw[7:0]
13	gz[4]
14	by[4]
22:15	gw[7:0]
23	bz[2]
24	gy[4]
32:25	bw[7:0]
33	bz[3]
34	bz[4]
40:35	rx[5:0]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	gz[1]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 7: (TWO) Red, Blue: 8-bit endpoint, 5-bit deltas

Green: 8-bit endpoint, 6-bit deltas

Bit	Description
4:0	mode = 10110
12:5	rw[7:0]
13	bz[0]

Bit	Description
14	by[4]
22:15	gw[7:0]
23	gy[5]
24	gy[4]
32:25	bw[7:0]
33	gz[5]
34	bz[4]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 8: (TWO) Red, Green: 8-bit endpoint, 5-bit deltas

Blue: 8-bit endpoint, 6-bit deltas

Bit	Description
4:0	mode = 11010
12:5	rw[7:0]
13	bz[1]
14	by[4]
22:15	gw[7:0]
23	by[5]
24	gy[4]
32:25	bw[7:0]
33	bz[5]
34	bz[4]
39:35	rx[4:0]

Bit	Description
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
60:55	bx[5:0]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 9: (TWO) Red, Green, Blue: 6-bit endpoints for all four, no deltas

Bit	Description
4:0	mode = 11110
10:5	rw[5:0]
11	gz[4]
12	bz[0]
13	bz[1]
14	by[4]
20:15	gw[5:0]
21	gy[5]
22	by[5]
23	bz[2]
24	gy[4]
30:25	bw[5:0]
31	gz[5]
32	bz[3]
33	bz[5]
34	bz[4]
40:35	rx[5:0]
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]

Bit	Description
60:55	bx[5:0]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 10: (ONE) Red, Green, Blue: 10-bit endpoints for both, no deltas

Bit	Description
4:0	mode = 00011
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
44:35	rx[9:0]
54:45	gx[9:0]
64:55	bx[9:0]
127:65	indices

Mode 11: (ONE) Red, Green, Blue: 11-bit endpoints, 9-bit deltas

Bit	Description
4:0	mode = 00111
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
43:35	rx[8:0]
44	rw[10]
53:45	gx[8:0]
54	gw[10]
63:55	bx[8:0]
64	bw[10]
127:65	indices

Mode 12: (ONE) Red, Green, Blue: 12-bit endpoints, 8-bit deltas

Bit	Description
4:0	mode = 01011
14:5	rw[9:0]
24:15	gw[9:0]

Bit	Description
34:25	bw[9:0]
42:35	rx[7:0]
43	rw[11]
44	rw[10]
52:45	gx[7:0]
53	gw[11]
54	gw[10]
62:55	bx[7:0]
63	bw[11]
64	bw[10]
127:65	indices

Mode 13: (ONE) Red, Green, Blue: 16-bit endpoints, 4-bit deltas

Bit	Description
4:0	mode = 01111
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[15]
40	rw[14]
41	rw[13]
42	rw[12]
43	rw[11]
44	rw[10]
48:45	gx[3:0]
49	gw[15]
50	gw[14]
51	gw[13]
52	gw[12]
53	gw[11]
54	gw[10]
58:55	bx[3:0]
59	bw[15]
60	bw[14]
61	bw[13]

Bit	Description
62	bw[12]
63	bw[11]
64	bw[10]
127:65	indices

Undefined mode values (10011, 10111, 11011, and 11111) return zero in the RGB channels.

The “indices” fields are defined as follows:

TWO mode *indices* field with fix-up index [1] at texel[3][3]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
89:87	texel[0][2] index
92:90	texel[0][3] index
95:93	texel[1][0] index
98:96	texel[1][1] index
101:99	texel[1][2] index
104:102	texel[1][3] index
107:105	texel[2][0] index
110:108	texel[2][1] index
113:111	texel[2][2] index
116:114	texel[2][3] index
119:117	texel[3][0] index
122:120	texel[3][1] index
125:123	texel[3][2] index
127:126	texel[3][3] index

TWO mode *indices* field with fix-up index [1] at texel[0][2]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
88:87	texel[0][2] index
91:89	texel[0][3] index
94:92	texel[1][0] index
97:95	texel[1][1] index
100:98	texel[1][2] index
103:101	texel[1][3] index

Bit	Description
106:104	texel[2][0] index
109:107	texel[2][1] index
112:110	texel[2][2] index
115:113	texel[2][3] index
118:116	texel[3][0] index
121:119	texel[3][1] index
124:122	texel[3][2] index
127:125	texel[3][3] index

TWO mode *indices* field with fix-up index [1] at texel[2][0]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
89:87	texel[0][2] index
92:90	texel[0][3] index
95:93	texel[1][0] index
98:96	texel[1][1] index
101:99	texel[1][2] index
104:102	texel[1][3] index
106:105	texel[2][0] index
109:107	texel[2][1] index
112:110	texel[2][2] index
115:113	texel[2][3] index
118:116	texel[3][0] index
121:119	texel[3][1] index
124:122	texel[3][2] index
127:125	texel[3][3] index

ONE mode *indices* field

Bit	Description
67:65	texel[0][0] index
71:68	texel[0][1] index
75:72	texel[0][2] index
79:76	texel[0][3] index
83:80	texel[1][0] index
87:84	texel[1][1] index
91:88	texel[1][2] index

Bit	Description
95:92	texel[1][3] index
99:96	texel[2][0] index
103:100	texel[2][1] index
107:104	texel[2][2] index
111:108	texel[2][3] index
115:112	texel[3][0] index
119:116	texel[3][1] index
123:120	texel[3][2] index
127:124	texel[3][3] index

Endpoint Computation

The endpoints can be defined in many different ways, as shown above. This section describes how the endpoints are computed from the bits in the compression block. The method used depends on whether the BC6H format is signed (BC6H_SF16) or unsigned (BC6H_UF16).

First, each channel (RGB) of each endpoint is extended to 16 bits. Each is handled identically and independently, however in some modes different channels have different incoming precision which must be accounted for. The following rules are employed:

- If the format is BC6H_SF16 or the endpoint is a delta value, the value is sign-extended to 16 bits
- For all other cases, the value is zero-extended to 16 bits

If there are no endpoints that are delta values, endpoint computation is complete. For endpoints that are delta values, the next step involves computing the absolute endpoint. The “w” endpoint is always absolute and acts as a base value for the other three endpoints. Each channel is handled identically and independently.

```
x = w + x
y = w + y
z = w + z
```

The above is performed using 16-bit integer arithmetic. Overflows beyond 16 bits are ignored (any resulting high bits are dropped).

Palette Color Computation

The next step involves computing the color palette values that provide the available values for each texel's color. The color palette for each line consists of the two endpoint colors plus 6 (TWO mode) or 14 (ONE mode) interpolated colors. Again each channel is processed independently.

First the endpoints are unquantized, with each channel of each endpoint being processed independently. The number of bits in the original base *w* value represents the precision of the endpoints. The input

endpoint is called *e*, and the resulting endpoints are represented as 17-bit signed integers and called e' below.

For the BC6H_UF16 format:

- if the precision is already 16 bits, e' = e
- if e = 0, e' = 0
- if e is the maximum representible in the precision, e' = 0xFFFF
- otherwise, e' = ((e « 16) + 0x8000) » precision

For the BC6H_SF16 format, the value is treated as sign magnitude. The sign is not changed, e' and e refer only to the magnitude portion:

- if the precision is already 16 bits, e' = e
- if e = 0, e' = 0
- if e is the maximum representible in the precision, e' = 0x7FFF
- otherwise, e' = ((e « 15) + 0x4000) » (precision - 1)

Next, the palette values are generated using predefined weights, using the tables below:

```
palette[i] = (w' * (64 - weight[i]) + x' * weight[i] + 32) » 6
```

TWO mode weights:

palette index	0	1	2	3	4	5	6	7
weight	0	9	18	27	37	46	55	64

ONE mode weights:

palette index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
weight	0	4	9	13	17	21	26	30	34	38	43	47	51	55	60	64

The two end palette indices are equal to the two endpoints given that the weights are 0 and 64. In the above equation w' and x' represent the endpoints e' computed in the previous step corresponding to w and x, respectively. For the second line in TWO mode, w and x are replaced with y and z.

The final step in computing the palette colors is to rescale the final results. For BC6H_UF16 format, the values are multiplied by 31/64. For BC6H_SF16, the values are multiplied by 31/32, treating them as sign magnitude. These final 16-bit results are ultimately treated as 16-bit floats.

Texel Selection

The final step is to select the appropriate palette index for each texel. This index then selects the 16-bit per channel palette value, which is re-interpreted as a 16-bit floating point result for input into the filter. This procedure differs depending on whether the mode is TWO or ONE.

ONE Mode

In ONE mode, there is only one set of palette colors, but the "indices" field is 63 bits. This field consists of a 4-bit palette index for each of the 16 texels, with the exception of the texel at [0][0] which has only 3 bits, the missing high bit being set to zero.

TWO Mode

32 partitions are defined for TWO, which are defined below. Each of the 32 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-1C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *w* and *x*) or line 1 (endpoints *y* and *z*). Each case has one texel each of "[0]" and "[1]", the index that this is at is termed the "fix-up index". These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	1	[0]	0	0	1	[0]	1	1	1	[0]	0	0	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	0	0	0	[1]	0	1	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	1	[0]	0	0	1	[0]	0	0	0
	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
08	[0]	0	0	0	[0]	0	1	1	[0]	0	0	0	[0]	0	0	0
	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0
	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
0C	[0]	0	0	1	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]
10	[0]	0	0	0	[0]	1	[1]	1	[0]	0	0	0	[0]	1	[1]	1
	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	1	1	1	0	0	0	0	0	[1]	0	0	0	0	0	0	1
	1	1	1	[1]	0	0	0	0	1	1	1	0	0	0	0	0
14	[0]	0	[1]	1	[0]	0	0	0	[0]	0	0	0	[0]	1	1	1
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1
	0	0	0	0	[1]	1	0	0	[1]	0	0	0	0	0	1	1
	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	[1]

18	[0]	0	[1]	1	[0]	0	0	0	[0]	1	[1]	0	[0]	0	[1]	1
	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	1	[1]	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0
1C	[0]	0	0	1	[0]	0	0	0	[0]	1	[1]	1	[0]	0	[1]	1
	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1
	[1]	1	1	0	[1]	1	1	1	1	0	0	0	1	0	0	1
	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0

The 46-bit “indices” field consists of a 3-bit palette index for each of the 16 texels, with the exception of the bracketed texels that have only two bits each. The high bit of these texels is set to zero.

BC7

These formats (BC7_UNORM and BC7_UNORM_SRGB) compresses 3-channel and 4-channel fixed point images.

The BC7 block is 16 bytes and represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel. BC7 has 8 different modes, the mode that the block is in is contained in the least significant bits (1-8 bits depending on mode).

The basic scheme consists of interpolating colors and alpha in some modes along either one, two, or three lines, with per-texel indices indicating which color/alpha along the line is chosen for each texel. If a two- or three-line mode is selected, one of 64 partition sets is indicated which selects which of the two lines each texel is assigned to, although some modes are limited to the first 16 partition sets. In the color-only modes, alpha is always returned at its default value of 1.0.

Some modes contain the following fields:

- **P-bits.** These represent shared LSB for all components of the endpoint, which increases the endpoint precision by one bit. In some cases both endpoints of a line share a P-bit.
- **Rotation bits.** For blocks with separate color and alpha, this 2-bit field allows selection of which of the four components has its own indexes (scalar) vs. the other three components (vector).
- **Index selector.** This 1-bit field selects whether the scalar or vector components uses the 3-bit index vs. the 2-bit index.

Field Definition

There are 8 possible modes for a BC7 block, the format of each is indicated in the 8 tables below. The mode is selected by the unique mode bits specified in each table. Each mode has particular characteristics described at the top of the table.

Mode 0: Color only, 3 lines (THREE), 4-bit endpoints with one P-bit per endpoint, 3-bit indices, 16 partitions

Bit	Description
0	mode = 0
4:1	partition
8:5	R0
12:9	R1
16:13	R2
20:17	R3
24:21	R4
28:25	R5
32:29	G0
36:33	G1
40:37	G2
44:41	G3
48:45	G4
52:49	G5
56:53	B0
60:57	B1
64:61	B2
68:65	B3
72:69	B4
76:73	B5
77	P0
78	P1
79	P2
80	P3
81	P4
82	P5
127:83	indices

Mode 1: Color only, 2 lines (TWO), 6-bit endpoints with one shared P-bit per line, 3-bit indices, 64 partitions

Bit	Description
1:0	mode = 10
7:2	partition
13:8	R0
19:14	R1
25:20	R2

Bit	Description
31:26	R3
37:32	G0
43:38	G1
49:44	G2
55:50	G3
61:56	B0
67:62	B1
73:68	B2
79:74	B3
80	P0
81	P1
127:82	indices

Mode 2: Color only, 3 lines (THREE), 5-bit endpoints, 2-bit indices, 64 partitions

Bit	Description
2:0	mode = 100
8:3	partition
13:9	R0
18:14	R1
23:19	R2
28:24	R3
33:29	R4
38:34	R5
43:39	G0
48:44	G1
53:49	G2
58:54	G3
63:59	G4
68:64	G5
73:69	B0
78:74	B1
83:79	B2
88:84	B3
93:89	B4
98:94	B5
127:99	indices

Mode 3: Color only, 2 lines (TWO), 7-bit endpoints with one P-bit per endpoint, 2-bit indices, 64 partitions

Bit	Description
3:0	mode = 1000
9:4	partition
16:10	R0
23:17	R1
30:24	R2
37:31	R3
44:38	G0
51:45	G1
58:52	G2
65:59	G3
72:66	B0
79:73	B1
86:80	B2
93:87	B3
94	P0
95	P1
96	P2
97	P3
127:98	indices

Mode 4: Color and alpha, 1 line (ONE), 5-bit color endpoints, 6-bit alpha endpoints, 16 2-bit indices, 16 3-bit indices, 2-bit component rotation, 1-bit index selector

Bit	Description
4:0	mode = 10000
6:5	rotation
7	index selector
12:8	R0
17:13	R1
22:18	G0
27:23	G1
32:28	B0
37:33	B1
43:38	A0
49:44	A1

Bit	Description
80:50	2-bit indices
127:81	3-bit indices

Mode 5: Color and alpha, 1 line (ONE), 7-bit color endpoints, 8-bit alpha endpoints, 2-bit color indices, 2-bit alpha indices, 2-bit component rotation

Bit	Description
5:0	mode = 100000
7:6	rotation
14:8	R0
21:15	R1
28:22	G0
35:29	G1
42:36	B0
49:43	B1
57:50	A0
65:58	A1
96:66	color indices
127:97	alpha indices

Mode 6: Combined color and alpha, 1 line (ONE), 7-bit endpoints with one P-bit per endpoint, 4-bit indices

Bit	Description
6:0	mode = 1000000
13:7	R0
20:14	R1
27:21	G0
34:28	G1
41:35	B0
48:42	B1
55:49	A0
62:56	A1
63	P0
64	P1
127:65	indices

Mode 7: Combined color and alpha, 2 lines (TWO), 5-bit endpoints with one P-bit per endpoint, 2-bit indices, 64 partitions

Bit	Description
7:0	mode = 10000000
13:8	partition
18:14	R0
23:19	R1
28:24	R2
33:29	R3
38:34	G0
43:39	G1
48:44	G2
53:49	G3
58:54	B0
63:59	B1
68:64	B2
73:69	B3
78:74	A0
83:79	A1
88:84	A2
93:89	A3
94	P0
95	P1
96	P2
97	P3
127:98	indices

Undefined mode values (bits 7:0 = 00000000) return zero in the RGB channels.

The indices fields are variable in length and due to the different locations of the fix-up indices depending on partition set there are a very large number of possible configurations. Each mode above indicates how many bits each index has, and the fix-up indices (one in ONE mode, two in TWO mode, and three in THREE mode) each have one less bit than indicated. However, the indices are always packed into the index fields according to the table below, with the specific bit assignments of each texel following the rules just given.

Bit	Description
LSBs	texel[0][0] index
	texel[0][1] index
	texel[0][2] index

Bit	Description
	texel[0][3] index
	texel[1][0] index
	texel[1][1] index
	texel[1][2] index
	texel[1][3] index
	texel[2][0] index
	texel[2][1] index
	texel[2][2] index
	texel[2][3] index
	texel[3][0] index
	texel[3][1] index
	texel[3][2] index
MSBs	texel[3][3] index

Endpoint Computation

The endpoints can be defined with different precision depending on mode, as shown above. This section describes how the endpoints are computed from the bits in the compression block. Each component of each endpoint follows the same steps.

If a P-bit is defined for the endpoint, it is first added as an additional LSB at the bottom of the endpoint value. The endpoint is then bit-replicated to create an 8-bit fixed point endpoint value with a range from 0x00 to 0xFF.

Palette Color Computation

The next step involves computing the color palette values that provide the available values for each texel's color. The color palette for each line consists of the two endpoint colors plus 2, 6, or 14 interpolated colors, depending on the number of bits in the indices. Again each channel is processed independently.

The equation to compute each palette color with index i, given two endpoints is as follows, using the tables below to determine the weight for each palette index:

```
palette[i] = (E0 * (64 - weight[i]) + E1 * weight[i] + 32) » 6
```

2-bit index weights:

palette index	0	1	2	3
weight	0	21	43	64

3-bit index weights:

palette index	0	1	2	3	4	5	6	7
weight	0	9	18	27	37	46	55	64

4-bit index weights:

palette index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
weight	0	4	9	13	17	21	26	30	34	38	43	47	51	55	60	64

The two end palette indices are equal to the two endpoints given that the weights are 0 and 64. In the above equation E0 and E1 represent the even-numbered and odd-numbered endpoints computed in the previous step for the component and line currently being computed.

Texel Selection

The final step is to select the appropriate palette index for each texel. This index then selects the 8-bit per channel palette value, which is interpreted as an 8-bit UNORM value for input into the filter (In BC7_UNORM_SRGB to UNORM values first go through inverse gamma conversion). This procedure differs depending on whether the mode is ONE, TWO, or THREE.

ONE Mode

In ONE mode, there is only one set of palette colors, thus there is only a single “partition set” defined, with all texels selecting line 0 and texel [0][0] being the “fix-up index” with one less bit in the index.

TWO Mode

64 partitions are defined for TWO, which are defined below. Each of the 64 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-3C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *0* and *1*) or line 1 (endpoints *2* and *3*). Each case has one texel each of “[0]” and “[1]”, the index that this is at is termed the “fix-up index”. These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	1	[0]	0	0	1	[0]	1	1	1	[0]	0	0	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	0	0	0	[1]	0	1	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	1	[0]	0	0	1	[0]	0	0	0
	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
08	[0]	0	0	0	[0]	0	1	1	[0]	0	0	0	[0]	0	0	0

	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0
	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
0C	[0]	0	0	1	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]
10	[0]	0	0	0	[0]	1	[1]	1	[0]	0	0	0	[0]	1	[1]	1
	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	1	1	1	0	0	0	0	0	[1]	0	0	0	0	0	0	1
	1	1	1	[1]	0	0	0	0	1	1	1	0	0	0	0	0
14	[0]	0	[1]	1	[0]	0	0	0	[0]	0	0	0	[0]	1	1	1
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1
	0	0	0	0	[1]	1	0	0	[1]	0	0	0	0	0	1	1
	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	[1]
18	[0]	0	[1]	1	[0]	0	0	0	[0]	1	[1]	0	[0]	0	[1]	1
	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	1	[1]	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0
1C	[0]	0	0	1	[0]	0	0	0	[0]	1	[1]	1	[0]	0	[1]	1
	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1
	[1]	1	1	0	[1]	1	1	1	1	0	0	0	1	0	0	1
	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0
20	[0]	1	0	1	[0]	0	0	0	[0]	1	0	1	[0]	0	1	1
	0	1	0	1	1	1	1	1	1	0	[1]	0	0	0	1	1
	0	1	0	1	0	0	0	0	0	1	0	1	[1]	1	0	0
	0	1	0	[1]	1	1	1	[1]	1	0	1	0	1	1	0	0
24	[0]	0	[1]	1	[0]	1	0	1	[0]	1	1	0	[0]	1	0	1
	1	1	0	0	0	1	0	1	1	0	0	1	1	0	1	0
	0	0	1	1	[1]	0	1	0	0	1	1	0	1	0	1	0
	1	1	0	0	1	0	1	0	1	0	0	[1]	0	1	0	[1]
28	[0]	1	[1]	1	[0]	0	0	1	[0]	0	[1]	1	[0]	0	[1]	1
	0	0	1	1	0	0	1	1	0	0	1	0	1	0	1	1
	1	1	0	0	[1]	1	0	0	0	1	0	0	1	1	0	1
	1	1	1	0	1	0	0	0	1	1	0	0	1	1	0	0
2C	[0]	1	[1]	0	[0]	0	1	1	[0]	1	1	0	[0]	0	0	0
	1	0	0	1	1	1	0	0	0	1	1	0	0	1	[1]	0

	1	0	0	1	1	1	0	0	1	0	0	1	0	1	1	0
	0	1	1	0	0	0	1	[1]	1	0	0	[1]	0	0	0	0
30	[0]	1	0	0	[0]	0	[1]	0	[0]	0	0	0	[0]	0	0	0
	1	1	[1]	0	0	1	1	1	0	0	[1]	0	0	1	0	0
	0	1	0	0	0	0	1	0	0	1	1	1	[1]	1	1	0
	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
34	[0]	1	1	0	[0]	0	1	1	[0]	1	[1]	0	[0]	0	[1]	1
	1	1	0	0	0	1	1	0	0	0	1	1	1	0	0	1
	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	0
	0	0	1	[1]	1	0	0	[1]	1	1	0	0	0	1	1	0
38	[0]	1	1	0	[0]	1	1	0	[0]	1	1	1	[0]	0	0	1
	1	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0
	1	1	0	0	0	0	1	1	1	0	0	0	1	1	1	0
	1	0	0	[1]	1	0	0	[1]	0	0	0	[1]	0	1	1	[1]
3C	[0]	0	0	0	[0]	0	[1]	1	[0]	0	[1]	0	[0]	1	0	0
	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0
	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1
	0	0	1	[1]	0	0	0	0	1	1	1	0	0	1	1	[1]

THREE Mode

64 partitions are defined for THREE, which are defined below. Each of the 64 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-3C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *0* and *1*), line 1 (endpoints *2* and *3*), or line 2 (endpoints *4* and *5*). Each case has one texel each of "[0]", "[1]", and "[2]", the index that this is at is termed the "fix-up index". These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	[1]	[0]	0	0	[1]	[0]	0	0	0	[0]	2	2	[2]
	0	0	1	1	0	0	1	1	2	0	0	1	0	0	2	2
	0	2	2	1	[2]	2	1	1	[2]	2	1	1	0	0	1	1
	2	2	2	[2]	2	2	2	1	2	2	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	[1]	[0]	0	2	[2]	[0]	0	1	1
	0	0	0	0	0	0	1	1	0	0	2	2	0	0	1	1
	[1]	1	2	2	0	0	2	2	1	1	1	1	[2]	2	1	1
	1	1	2	[2]	0	0	2	[2]	1	1	1	[1]	2	2	1	[1]
08	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0	[0]	0	1	2
	0	0	0	0	1	1	1	1	1	1	[1]	1	0	0	[1]	2

	[1]	1	1	1	[1]	1	1	1	2	2	2	2	0	0	1	2
	2	2	2	[2]	2	2	2	[2]	2	2	2	[2]	0	0	1	[2]
0C	[0]	1	1	2	[0]	1	2	2	[0]	0	1	[1]	[0]	0	1	[1]
	0	1	[1]	2	0	[1]	2	2	0	1	1	2	2	0	0	1
	0	1	1	2	0	1	2	2	1	1	2	2	[2]	2	0	0
	0	1	1	[2]	0	1	2	[2]	1	2	2	[2]	2	2	2	0
10	[0]	0	0	[1]	[0]	1	1	[1]	[0]	0	0	0	[0]	0	2	[2]
	0	0	1	1	0	0	1	1	1	1	2	2	0	0	2	2
	0	1	1	2	[2]	0	0	1	[1]	1	2	2	0	0	2	2
	1	1	2	[2]	2	2	0	0	1	1	2	[2]	1	1	1	[1]
14	[0]	1	1	[1]	[0]	0	0	[1]	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	1	0	0	[1]	1	1	1	0	0
	0	2	2	2	[2]	2	2	1	0	1	2	2	[2]	2	[1]	0
	0	2	2	[2]	2	2	2	1	0	1	2	[2]	2	2	1	0
18	[0]	1	2	[2]	[0]	0	1	2	[0]	1	1	0	[0]	0	0	0
	0	[1]	2	2	0	0	1	2	1	2	[2]	1	0	1	[1]	0
	0	0	1	1	[1]	1	2	2	[1]	2	2	1	1	2	[2]	1
	0	0	0	0	2	2	2	[2]	0	1	1	0	1	2	2	1
1C	[0]	0	2	2	[0]	1	1	0	[0]	0	1	1	[0]	0	0	0
	1	1	0	2	0	[1]	1	0	0	1	2	2	2	0	0	0
	[1]	1	0	2	2	0	0	2	0	1	[2]	2	[2]	2	1	1
	0	0	2	[2]	2	2	2	[2]	0	0	1	[1]	2	2	2	[1]
20	[0]	0	0	0	[0]	2	2	[2]	[0]	0	1	[1]	[0]	1	2	0
	0	0	0	2	0	0	2	2	0	0	1	2	0	[1]	2	0
	[1]	1	2	2	0	0	1	2	0	0	2	2	0	1	[2]	0
	1	2	2	[2]	0	0	1	[1]	0	2	2	[2]	0	1	2	0
24	[0]	0	0	0	[0]	1	2	0	[0]	1	2	0	[0]	0	1	1
	1	1	[1]	1	1	2	0	1	2	0	1	2	2	2	0	0
	2	2	[2]	2	[2]	0	[1]	2	[1]	[2]	0	1	1	1	[2]	2
	0	0	0	0	0	1	2	0	0	1	2	0	0	0	1	[1]
28	[0]	0	1	1	[0]	1	0	[1]	[0]	0	0	0	[0]	0	2	2
	1	1	[2]	2	0	1	0	1	0	0	0	0	1	[1]	2	2
	2	2	0	0	2	2	2	2	[2]	1	2	1	0	0	2	2
	0	0	1	[1]	2	2	2	[2]	2	1	2	[1]	1	1	2	[2]
2C	[0]	0	2	[2]	[0]	2	2	0	[0]	1	0	1	[0]	0	0	0
	0	0	1	1	1	2	[2]	1	2	2	[2]	2	2	1	2	1
	0	0	2	2	0	2	2	0	2	2	2	2	[2]	1	2	1

	0	0	1	[1]	1	2	2	[1]	0	1	0	[1]	2	1	2	[1]
30	[0]	1	0	[1]	[0]	2	2	[2]	[0]	0	0	2	[0]	0	0	0
	0	1	0	1	0	1	1	1	1	[1]	1	2	2	[1]	1	2
	0	1	0	1	0	2	2	2	0	0	0	2	2	1	1	2
	2	2	2	[2]	0	1	1	[1]	1	1	1	[2]	2	1	1	[2]
34	[0]	2	2	2	[0]	0	0	2	[0]	1	1	0	[0]	0	0	0
	0	[1]	1	1	1	1	1	2	0	[1]	1	0	0	0	0	0
	0	1	1	1	[1]	1	1	2	0	1	1	0	2	1	[1]	2
	0	2	2	[2]	0	0	0	[2]	2	2	2	[2]	2	1	1	[2]
38	[0]	1	1	0	[0]	0	2	2	[0]	0	2	2	[0]	0	0	0
	0	[1]	1	0	0	0	1	1	1	1	2	2	0	0	0	0
	2	2	2	2	0	0	[1]	1	[1]	1	2	2	0	0	0	0
	2	2	2	[2]	0	0	2	[2]	0	0	2	[2]	2	[1]	1	[2]
3C	[0]	0	0	[2]	[0]	2	2	2	[0]	1	0	[1]	[0]	1	1	[1]
	0	0	0	1	1	2	2	2	2	2	2	2	2	0	1	1
	0	0	0	2	0	2	2	2	2	2	2	2	[2]	2	0	1
	0	0	0	[1]	[1]	2	2	[2]	2	2	2	[2]	2	2	2	0

Adaptive Scalable Texture Compression (ASTC)

This section describes the data structure of the Adaptive Scalable Texture Compression (ASTC) format, as well as the decoding flow of ASTC. Also described are the header format and mipmap layout in the compressed texture file of *.astc. This is based on the reference encoder and decoder from the Khronos committee, with an extension to support multiple miplevel texture.

ASTC is a new compressed texture format with following characteristics:

1. ASTC compression format is currently only used for static texture, due to the large amount of computation and high latency required to find the optimal configuration in compression. It cannot be used to compress dynamic textures such as a shadow map.
2. ASTC is a lossy compression technique that cannot be used to compress dynamic textures which do not tolerate quality degradation.
3. ASTC has a huge range of compression ratio and block size, but these choices are fixed for each texture for all blocks at all mipmap levels.
4. ASTC has options to support compression from 1 to 4 channels for texture data.
5. ASTC can support both high and low dynamic textures.
6. ASTC can support both 2D and 3D textures.

Supported Formats
2D LDR profile.
2D HDR profiles.

ASTC Fundamentals

This section describes some background details and new surface formats for ASTC.

Background

ASTC is a more advanced texture compression technique than the existing BC and ETC, and can reduce footprint & BW of static texture further in Graphics application by providing a texture compression solution at higher compression ratios. To best find the balance point of visual quality and compression, it provides a wide range of bit rate selection from 8bpp to 0.89 bpp in 2D, and 4.6bpp to 0.6 bpp in 3D at various block size of footprints. It also has flexibility to specify 1-4 components, selection of dual plane mode among the specified color components.

It extends the existing linear model on color distribution of each block in multiple partitions (up to 4), with flexible compact supporting on index/weight for color interpolation. ASTC also has a support of high dynamic range (HDR) image and 3D textures. The mixture of HDR and LDR data is within each block level allows a great flexibility to represent high dynamics variation at fine granularity. The support of 3D texture explores the data coherency in all 3 dimensions, without the need to mimic 3D map with 2D slices. On top of everything, void-extent regions are introduced for both 2D and 3D maps as further optimization on large constant region.

ASTC is a voted approved future texture format for OpenGL ES by Khronos Group, and is on projection to be accepted in D3D API. It provides less bandwidth, storage, lower power and high performance over existing techniques, and has been identified as a critical feature for Gen GPU to get future design win from desktop, laptop to tablet and handheld markets.

Due to the computational complexity and processing delay of the encoding process, ASTC compression encoding is always offline, and can only be used for static texture. It does not support auto mipmap generation and cannot be considered as a format for render target.

The ASTC provides a wide spectrum of bit per pixel for both 2D and 3D texture for both LDR and HDR images, hence a wide range of compression to any 2D and 3D texture.

LDR Compression Ratios:

2D Block Footprint	Bit Rate (bpp)	Compression ratio (LDR 32bpp)
4x4	8.00	4.0
5x4	6.40	5.0
5x5	5.12	6.3
6x5	4.27	7.5
6x6	3.56	9.0
8x5	3.20	10.0
8x6	2.67	12.0
10x5	2.56	12.5
10x6	2.13	15.0
8x8	2.00	16.0
10x8	1.60	20.0
10x10	1.28	25.0
12x10	1.07	29.9
12x12	0.89	36.0

HDR Compression Ratios:

2D Block Footprint	Bit Rate (bpp)	Compression ratio (LDR 32bpp)
4x4	8.00	8.0
5x4	6.40	10.0
5x5	5.12	12.5
6x5	4.27	15.0
6x6	3.56	18.0
8x5	3.20	20.0
8x6	2.67	24.0
10x5	2.56	25.0
10x6	2.13	30.0
8x8	2.00	32.0
10x8	1.60	40.0
10x10	1.28	50.0
12x10	1.07	59.8
12x12	0.89	71.9

Compared against fixed compression ratios of 4x or 8x on BC* formats, ASTC provides compression ratios from 4x to 36x for 2D LDR, 8x to 72x in 2D HDR maps, 7x to 54x on 3D LDR (32bpp) maps, and 14x to 108x in 3D HDR (64bpp). This can reduce bandwidth and footprint of a static 2D HDR or 3D textures to a small fractional of the existing BC formats, and greatly improve the performance on the graphic applications using these textures intensively.

Another benefit of ASTC is that, with the large range of selection of footprints and bpp, it can provide a good trade-off between quality degradation of the compressed texture and performance, due to the bandwidth and footprints reduction. This could not be achieved by any previously existing texture compression technologies.

Although ASTC has a huge benefit of bandwidth reduction, the expected performance gain in real 3D application from this technique depends on how much texture bandwidth bottleneck is relative to the throughput of computing in EU, Sampler, and other fixed function components.

New Surface Formats for ASTC Texture

The ASTC data format natively supports 14 2D block size, 10 3D block size, and each decoded format should support either UN8 (with sRGB con version) or Float16 at each color component. Following is the full list of all different surface formats as the full combination of different block shapes and UN8 or Float16 options.

Programming Note	
Context:	Supported ASTC Formats
All 2D ASTC Formats (LDR and HDR) are supported.	

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
000h	000	0	0		000 000 000	ASTC_LDR_2D_4x4_U8sRGB	8.00
008h	000	1	0		000 001 000	ASTC_LDR_2D_5x4_U8sRGB	6.40
009h	000	1	1		000 001 001	ASTC_LDR_2D_5x5_U8sRGB	5.12
011h	000	2	1		000 010 001	ASTC_LDR_2D_6x5_U8sRGB	4.27
012h	000	2	2		000 010 010	ASTC_LDR_2D_6x6_U8sRGB	3.56
021h	000	4	1		000 100 001	ASTC_LDR_2D_8x5_U8sRGB	3.20
022h	000	4	2		000 100 010	ASTC_LDR_2D_8x6_U8sRGB	2.67
031h	000	6	1		000 110 001	ASTC_LDR_2D_10x5_U8sRGB	2.56

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
032h	000	6	2		000 110 010	ASTC_LDR_2D_10x6_U8sRGB	2.13
024h	000	4	4		000 100 100	ASTC_LDR_2D_8x8_U8sRGB	2.00
034h	000	6	4		000 110 100	ASTC_LDR_2D_10x8_U8sRGB	1.60
036h	000	6	6		000 110 110	ASTC_LDR_2D_10x10_U8sRGB	1.28
03eh	000	7	6		000 111 110	ASTC_LDR_2D_12x10_U8sRGB	1.07
03fh	000	7	7		000 111 111	ASTC_LDR_2D_12x12_U8sRGB	0.89
040h	001	0	0		001 000 000	ASTC_LDR_2D_4x4_FLT16	8.00
048h	001	1	0		001 001 000	ASTC_LDR_2D_5x4_FLT16	6.40
049h	001	1	1		001 001 001	ASTC_LDR_2D_5x5_FLT16	5.12
051h	001	2	1		001 010 001	ASTC_LDR_2D_6x5_FLT16	4.27
052h	001	2	2		001 010 010	ASTC_LDR_2D_6x6_FLT16	3.56
061h	001	4	1		001 100 001	ASTC_LDR_2D_8x5_FLT16	3.20
062h	001	4	2		001 100 010	ASTC_LDR_2D_8x6_FLT16	2.67
071h	001	6	1		001 110 001	ASTC_LDR_2D_10x5_FLT16	2.56

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
072h	001	6	2		001 110 010	ASTC_LDR_2D_10x6_FLT16	2.13
064h	001	4	4		001 100 100	ASTC_LDR_2D_8x8_FLT16	2.00
074h	001	6	4		001 110 100	ASTC_LDR_2D_10x8_FLT16	1.60
076h	001	6	6		001 110 110	ASTC_LDR_2D_10x10_FLT16	1.28
07eh	001	7	6		001 111 110	ASTC_LDR_2D_12x10_FLT16	1.07
07fh	001	7	7		001 111 111	ASTC_LDR_2D_12x12_FLT16	0.89
080h	010	0	0	0	010 000 000	ASTC_LDR_3D_3x3x3_U8sRGB	4.74
090h	010	1	0	0	010 010 000	ASTC_LDR_3D_4x3x3_U8sRGB	3.56
094h	010	1	1	0	010 010 100	ASTC_LDR_3D_4x4x3_U8sRGB	2.67
095h	010	1	1	1	010 010 101	ASTC_LDR_3D_4x4x4_U8sRGB	2.00
0a5h	010	2	1	1	010 100 101	ASTC_LDR_3D_5x4x4_U8sRGB	1.60
0a9h	010	2	2	1	010 101 001	ASTC_LDR_3D_5x5x4_U8sRGB	1.28
0aah	010	2	2	2	010 101 010	ASTC_LDR_3D_5x5x5_U8sRGB	1.02
0bah	010	3	2	2	010 111 010	ASTC_LDR_3D_6x5x5_U8sRGB	0.85

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
0beh	010	3	3	2	010 111 110	ASTC_LDR_3D_6x6x5_U8sRGB	0.71
0bfh	010	3	3	3	010 111 111	ASTC_LDR_3D_6x6x6_U8sRGB	0.59
140h	101	0	0	n/a	101 000 000	ASTC_FULL_2D_4x4_FLT16	8.00
148h	101	1	0	n/a	101 001 000	ASTC_FULL_2D_5x4_FLT16	6.40
149h	101	1	1	n/a	101 001 001	ASTC_FULL_2D_5x5_FLT16	5.12
151h	101	2	1	n/a	101 010 001	ASTC_FULL_2D_6x5_FLT16	4.27
152h	101	2	2	n/a	101 010 010	ASTC_FULL_2D_6x6_FLT16	3.56
161h	101	4	1	n/a	101 100 001	ASTC_FULL_2D_8x5_FLT16	3.20
162h	101	4	2	n/a	101 100 010	ASTC_FULL_2D_8x6_FLT16	2.67
171h	101	6	1	n/a	101 110 001	ASTC_FULL_2D_10x5_FLT16	2.56
172h	101	6	2	n/a	101 110 010	ASTC_FULL_2D_10x6_FLT16	2.13
164h	101	4	4	n/a	101 100 100	ASTC_FULL_2D_8x8_FLT16	2.00
174h	101	6	4	n/a	101 110 100	ASTC_FULL_2D_10x8_FLT16	1.60
176h	101	6	6	n/a	101 110 110	ASTC_FULL_2D_10x10_FLT16	1.28

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
17eh	101	7	6	n/a	101 111 110	ASTC_FULL_2D_12x10_FLT16	1.07
17fh	101	7	7	n/a	101 111 111	ASTC_FULL_2D_12x12_FLT16	0.89
1c0h	111	0	0	0	111 000 000	ASTC_FULL_3D_3x3x3_FLT16	4.74
1d0h	111	1	0	0	111 010 000	ASTC_FULL_3D_4x3x3_FLT16	3.56
1d4h	111	1	1	0	111 010 100	ASTC_FULL_3D_4x4x3_FLT16	2.67
1d5h	111	1	1	1	111 010 101	ASTC_FULL_3D_4x4x4_FLT16	2.00
1e5h	111	2	1	1	111 100 101	ASTC_FULL_3D_5x4x4_FLT16	1.60
1e9h	111	2	2	1	111 101 001	ASTC_FULL_3D_5x5x4_FLT16	1.28
1eah	111	2	2	2	111 101 010	ASTC_FULL_3D_5x5x5_FLT16	1.02
1fah	111	3	2	2	111 111 010	ASTC_FULL_3D_6x5x5_FLT16	0.85
1feh	111	3	3	2	111 111 110	ASTC_FULL_3D_6x6x5_FLT16	0.71
1ffh	111	3	3	3	111 111 111	ASTC_FULL_3D_6x6x6_FLT16	0.59

ASTC File Format and Memory Layout

ASTC Header Data Structure and Amendment

The 1st block of an ASTC compression texture is a header file. Its byte layout in the original header structure in *.astc file is:

```
struct astc_header
{
    uint8_t magic[4];
    uint8_t blockdim_x;
    uint8_t blockdim_y;
    uint8_t blockdim_z;
    uint8_t xsize[3]; // x-size = xsize[0] + xsize[1] + xsize[2]
    uint8_t ysize[3]; // x-size, y-size and z-size are given in texels;
    uint8_t zsize[3]; // block count is inferred
};
```

Since there are limited ranges for block dimensions in x, y and z directions as described in following, we could store additional information in the unused upper bits of these byte fields

Block Dimension	2D	3D
blockdim_x	4, 5, 6, 8, 10, 12	3, 4, 5, 6
blockdim_y	4, 5, 6, 8, 10, 12	3, 4, 5, 6
blockdim_z	1	3, 4, 5, 6

Since blockdim_z is in the range of [1,6], only lower 3 bits of blockdim_z is used. We proposed the **Intel astc extension format** with *numLODs* stored in the upper 5 bits of the byte field used for blockdim_z. This new byte field can be defined as:

numLODs_blockdim_z = (numLODs-1) « 3 | (blockdim_z & 0x7) ;

New header:

```
struct astc_header
{
    uint8_t magic[4];
    uint8_t blockdim_x;
    uint8_t blockdim_y;
uint8_t numLODs_blockdim_z;
uint8_t xsize[3]; // width = xsize[0] + (xsize[1]«8) + (xsize[2]«16)
uint8_t ysize[3]; // height= ysize[0] + (ysize[1]«8) + (ysize[2]«16)
uint8_t zsize[3]; // depth = zsize[0] + (zsize[1]«8) + (zsize[2]«16)
// x_size, y_size and z_size are given in texels;

// block count is inferred
};
```

The driver or the software responsible for managing the memory resource will get numLODs and blockdim_z in:

numLODs = ((numLODs_blockdim_z » 3) & 0x1F) + 1;

blockdim_z = numLODs_blockdim_z & 0x7;

Data Layout in ASTC Compression File

A number of parameters are useful to determine where given pixels are located on the 2D & 3D surface. First, the width and height for each LOD level "L" is computed as:

$$W_L = \big((width \gg L) > 0\big)?\, width \gg L : 1\big)$$

$$H_L = \big((height \gg L) > 0\big)?\, height \gg L : 1\big)$$

$$D_L = \big((depth \gg L) > 0\big)?\, depth \gg L : 1\big)$$

The numbers of blocks in width, height and depth slab in each LOD are:

Nw(L) = Ceil(WL /Bw);

Nh (L) = Ceil(HL /Bh);

Ns (L) = Ceil(DL /Bd),

Where Bw, Bh and Bd is the block width, height and depth respectively.

Since ASTC has a native tile format specified by the encoding block size, the total number of blocks in each LOD level of the mipmap is described by nBL = Nw(L) * Nh (L) * Ns (L), The total number of blocks in the entire texture map is a summation of nBL's from all mipmap levels and all slabs, which are all pre-compressed via ASTC encoder. All the blocks in each LOD are in raster sequenced in width, height and then depth slab order.

Total ASTC Data Block Layout in All Mipmap Levels

The entire layout of the compression texture file looks like:

Address	Data Description
Addr0 (Base Address)	Header structure
Addr0+16	1st Data Block in LOD_0
Addr0+32	2nd Data Block in LOD_0
…	…
Addr1 = Addr0+16*nB0	Last Data Block in LOD_0

Address	Data Description
Addr1 +16	1st Data Block in LOD_1
Addr1+32	2nd Data Block in LOD_1
…	…
Addr2 = Addr1+16*nB1	Last Data Block in LOD_1
Addr2+16	1st Data Block in LOD_2
Addr2+32	2nd Data Block in LOD_2
…	…
Addr3 = Addr2+16*nB2	Last Data Block in LOD_2
…	…

Data Layout in Memory for All Mipmap Levels

The following equations for give the base address (U_offset, V_offset) in Cartesian coordinates for the starting point of each mip map at LOD L and depth slab q:

LOD=0:

U_offset (0, q) = 0;

*V_offset (0, q) = q * h0;*

LOD=1:

*U_offset (1, q) = (q%2)*w1;*

*V_offset (1, q) = D0*h0 + (q»1)*h1;*

LOD=2:

*U_offset (2, q) = (q%4)*w2;*

*V_offset (2, q) = D0*h0 + ceil(D1/2) * h1 + (q»2)*h2;*

LOD=3:

*U_offset (3, q) = (q%8)*w3;*

*V_offset (3, q) = D0*h0 + ceil(D1/2) * h1 + ceil(D2/2) * h2 + (q»3)*h3;*

••••••

Since ASTC has a native tile format specified by the encoding block size, the total number of blocks in each LOD level of the mipmap is described by nBL = Nw(L) * Nh (L) * Ns (L). The memory layout for TileY format are considered with 512bit (16Bx4) in 1 cacheline granularity, the total number of blocks is: 4*((Ceil(HL /Bh)+3)/4 * Ceil(WL /Bw) * Ceil(DL /Bd):

Here is the full list describing the total number of rows and columns of data in each mipmap for texture in ASTC format:

Table for block dimension in 2D

Block Size	ASTC Block Height (in line)	ASTC Block Width (in Byte)
4	((Ceil(HL /4) +3)/4) *4	Ceil(WL /4) * 16
5	((Ceil(HL /5) +3)/4) *4	Ceil(WL /5) * 16
6	((Ceil(HL /6) +3)/4) *4	Ceil(WL /6) * 16
8	((Ceil(HL /8) +3)/4) *4	Ceil(WL /8) * 16
10	((Ceil(HL /10) +3)/4) *4	Ceil(WL /10) * 16
12	((Ceil(HL /12) +3)/4) *4	Ceil(WL /12) * 16

Table for block dimension in 3D

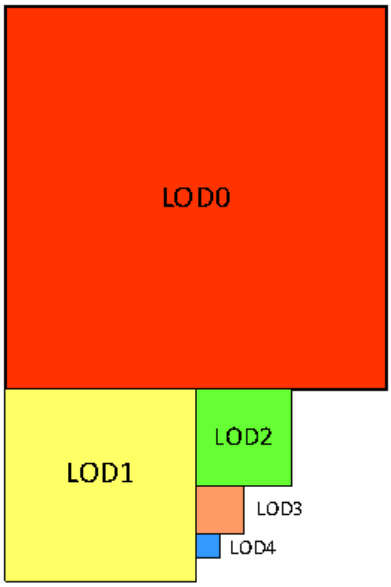
Block Size	ASTC Block Height (in line)	ASTC Block Width (in Byte)	ASTC Block Depth/slab (in slice)
3	((Ceil(HL /3) +3)/4) *4	Ceil(WL /3) * 16	Ceil(DL /3)
4	((Ceil(HL /4) +3)/4) *4	Ceil(WL /4) * 16	Ceil(DL /4)
5	((Ceil(HL /5) +3)/4) *4	Ceil(WL /5) * 16	Ceil(DL /5)
6	((Ceil(HL /6) +3)/4) *4	Ceil(WL /6) * 16	Ceil(DL /6)

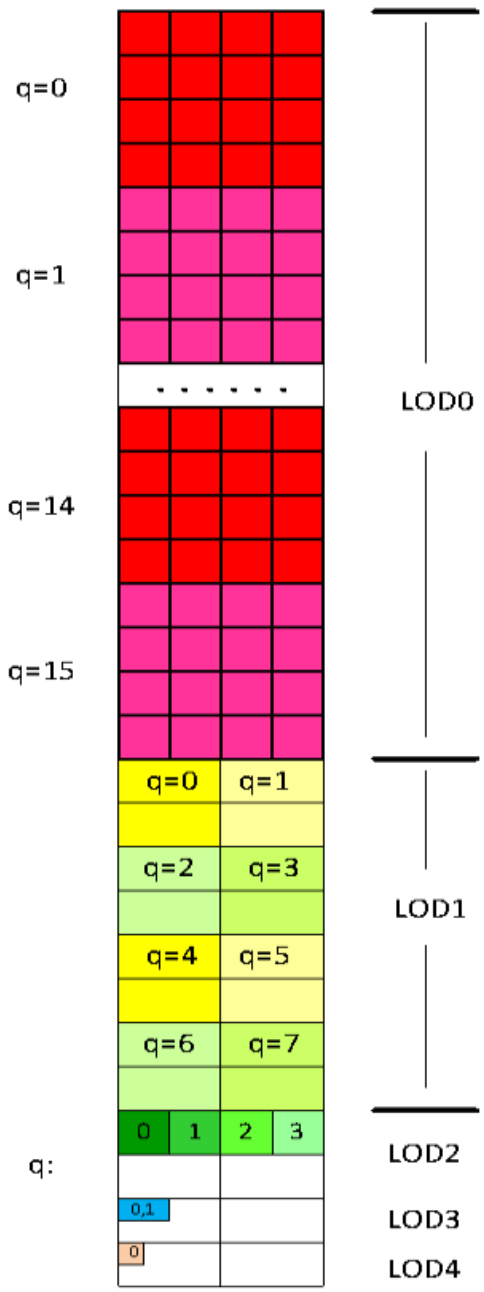
For example, an image of 64x64 with 5x5 block coding in LOD0 will have:

Block Height: (13+3)/4*4=16 (lines)

Block Width: 13 *16 = 208 (Bytes)

The following diagram illustrate the memory layout for 2D and 3D map respectively.


LOD0
LOD1
LOD2
LOD3
LOD4



ASTC Data Structure

Layout and Description of Block Data

The block data structure is described in the following table in the categories of the block being partition enabled (2-4 partitions) or disabled (only 1 partition), as well as 1 plane or dual-plane mode. Where CEM refers to Color Endpoint Mode, and CCS stands for Color Channel Selection:

Layout of Partitioning Disabled (1 partition) and Enabled (multi-partition) blocks

Partition Disable	127:19					18:17	16:13	12:11	10:0
(1 Plane)	Index Data				Color Endpoint Data		CEM	00	Index Mode
(2 Planes)	Index Data				Color Endpoint Data	CCS	CEM	00	Index Mode
Multi-partitions	127:29					28:23	22:13	12:11	10:0
(1 Plane)	Index Data	Rest of CEM			Color Endpoint Data	CEM (Initial 6 bits)	Partition Index	Part	Index Mode
(2 Planes)	Index Data	CCS	Rest of CEM		Color Endpoint Data	CEM (Initial 6 bits)	Partition Index	Part	Index Mode

The 11 bit “Index mode” field specifies how the Texel Index Data is encoded. The bit encoding of this field is listed in next two tables, one for the 2D and one for the 3D.

The “Part” field specifies the number of partitions minus one. If dual plane mode is enabled, the number of partitions must be 3 or fewer. In case 4 partitions in such situation are specified, the error value is returned for all texels in the block. The size and layout of the extra configuration data depends on the number of partition, and the number of planes in the image.

Partitioning

For any non-void extend region, each block is subdivided into 1, 2, 3 or 4 partitions, with a separate color endpoint pair for each partition. The number of partitions is specified by the partition count-1 in bits [12:11] of block data. If 2 or more partitions are selected, partitioning is enabled, the 10 bit partition index is then used to select one from 1024 partitioning patterns, where the total set of patterns supported in ASTC depends on the partition count and block size. The partitioning patterns are produced generatively, which supports a very large set of partitioning patterns for different block sizes with a modest number of hardware gates implementation.

Index Mode

The “Index mode” field specifies how the Texel Index Data is encoded. The bit encoding of this field is listed in next two tables, one for the 2D and one for the 3D.

The Index Mode field specifies the width (N), height (M) and depth (Q) of the grid of indices, what range of values they use, and whether dual index planes are present. The index ranges are encoded using a 3 bit value R, which is interpreted together with a precision bit H, as follows:

Mode R(r2 r1 r0)	Low-precision (H=0)					High-precision (H=1)			
	Index Range	Trits	Quints	Bits		Index Range	Trits	Quints	Bits
0 0 0	Invalid					Invalid			
0 0 1	Invalid					Invalid			
0 1 0	[0,1]			1		[0,9]		1	1
0 1 1	[0,2]	1				[0,11]	1		2
1 0 0	[0,3]			2		[0,15]			4
1 0 1	[0,4]		1			[0,19]		1	2
1 1 0	[0,5]	1		1		[0,23]	1		3
1 1 1	[0,7]			3		[0,31]			5

Each index value is encoded using the specified number of Trits, Quints and Bits. The details of this encoding can be found in Section - *Integer Sequence Encoding*. Due to the encoding of the R field, bits r2 and r1 cannot both be zero,

The number of indices provided for a block is not tied to the block size in any way, instead, the indices form an N*M*Q ordered grid. N, M and Q are specified on a per-block basis rather then being a global texture property. For 2D blocks, N and M can be set to any value from 2 to 12 while Q is fixed at 1; for 3D blocks, N, M and Q can be set to any value from 2 to 5. The range used for each index can be set separately for each block. The Index Bit Mode field species the values of N, M, Q and the range; it also specifies whether Dual Index Planes are present or not as well.

The D bit in following tables is set to indicate dual-plane mode. In this mode, the maximum allowed number of partitions is 3. The size of the grid in each dimension must be less than or equal to the corresponding dimension of the block footprint. If the grid size is greater than the footprint dimension in any axis, then this is an illegal block encoding and all texels will decode to the error color.

For 2D blocks, the index mode field is laid out as follows:

The bit encoding of the index mode field for 2D Blocks

Bits									Width	Height	Notes
10	9	8	7	6	5	4	3:2	1:0	N	M	
D	H	B		A		r0	0 0	r2 r1	B+4	A+2	

Bits											Width	Height	Notes
D	H	B		A		r0	0 1	r2 r1			B+8	A+2	
D	H	B		A		r0	1 0	r2 r1			A+2	B+8	
D	H	0	B	A		r0	1 1	r2 r1			A+2	B+6	
D	H	1	B	A		r0	1 1	r2 r1			B+2	A+2	
D	H	0	0	A		r0	r2 r1	0 0			12	A+2	
D	H	0	1	A		r0	r2 r1	0 0			A+2	12	
D	H	1	1	0	0	r0	r2 r1	0 0			6	10	
D	H	1	1	0	1	r0	r2 r1	0 0			10	6	
B		1	0	A		r0	r2 r1	0 0			A+6	B+6	D=0, H=0
x	x	1	1	1	1	1	1 1	0 0			-	-	Void-Extent
x	x	1	1	1	x	x	x x	0 0			-	-	Reserve
x	x	x	x	x	x	x	0 0	0 0			-	-	Reserve

Note that, due to the encoding of the R field (r0, r1, r2), bits r2 and r1 cannot both be zero, which disambiguates the first five rows from the rest of the table. The penultimate row of the table is reserved only if bits [5:2] are not all 1, in which case it encodes a void-extent block (as shown in the previous row)

For 3D blocks, the index mode field is laid out as follows:

3D Index Mode Layout

Bits											Notes			
10	9	8	7	6	5	4	3	2	1	0	N	M	Q	(D, H)
D	H	B		A		r0	C		r2 r1		A+2	B+2	C+2	
B		0	0	A		r0	r2 r1		0	0	6	B+2	A+2	(0, 0)
B		0	1	A		r0	r2 r1		0	0	A+2	6	B+2	(0, 0)
B		1	0	A		r0	r2 r1		0	0	A+2	B+2	6	(0, 0)

Bits										Notes			
D	H	1	1	0	0	r0	r2 r1	0	0	6	2	2	
D	H	1	1	0	1	r0	r2 r1	0	0	2	6	2	
D	H	1	1	1	0	r0	r2 r1	0	0	2	2	6	
x	x	1	1	1	1	1	1 1	0	0	-	-	-	Void-Extent
x	x	1	1	1	1	x	x x	0	0	-	-	-	Reserve
x	x	x	x	x	x	x	0 0	0	0	-	-	-	Reserve

The D bit is set to indicate dual-plane mode:

1: dual index planes are used

0: single index plane is used

In this mode, the maximum allowed number of partitions is 3. The size of the grid in each dimension must be less than or equal to the corresponding dimension of the block footprint. If the grid size is greater than the footprint dimension in any axis, then this is an illegal block encoding and all texels will decode to the error color. The penultimate row of the table is reserved only if bits [4:2] are not all 1, in which case it encodes a void-extent block (as shown in the previous row).

H: Index Range Bit:

1: the High-Precision group is selected.

0: The Low-Precision group is selected.

Here is the detail description:

- The encoding of xx111111100 is for the void-extent block.
- The pattern xxxxxxx0000 (the bottom 4 bits being 0000b) is reserved for future extension, and should result a NaN-vector when such a pattern is decoded.
- Any encodings not listed in the table are considered invalid and result in undened behavior if encountered by decoders.

Given the limitation of the fix length of 128 bits per block, there are restrictions that will not allow every possible encoding:

- The total number of indexes (N*M*Q for single index plane, 2*N*M*Q for dual index planes) must not exceed 64.
- The length of the Index Integer Sequence must not exceed 96 bits.
- The length of the Index Integer Sequence must be at least 24 bits.

- The above restriction, combined with the other field widths of the format, implicitly restricts the Color Integer Sequence to a maximum of 75 bits.
- Blocks that violate these restrictions are not legally produced by the encoder, result a vector of NaNs if encountered by decoders.

Here is how the indices in each block are encoded and stored:

- They are encoded using the Integer Sequence Encoding method described in Appendix.
- The resulting bit-sequence is then bit-reversed, and stored from the top of the block downwards. The ordering of the indices in the Integer Sequence is a simple scan line-like ordering.

The indices are used in two steps to interpolate between two endpoint colors for each texel.

- First, they are scaled from whatever interval they were to the range [0,64];
- The resulting value is then used as a weight to interpolate between the two endpoints.

Index Planes

Depending on the Index Bits mode selected, an ASTC compressed block may offer 1 or 2 index planes. In the case of 2 index planes, two indices rather than just one are supplied for each texel that receives indices. Of these two indices, the first one is used for a weighted sum of three of the color components; the second is used for a weighted sum of the fourth color component. If only 1 index plane is present, it applies to all four color components.

If two index planes are used, then a 2-bit bit field is needed to indicate which of the color components the second index plane applies to. These two bits are stored just below the index bits, except in the case where leftover color endpoint type bits are present; in that case, these two bits are stored just below the leftover color endpoint type bits. This two-bit bit-field has the following layout:

Channel	Red	Green	Blue	Alpha
Value	0	1	2	3

If index infill is present while two index planes are being used, then index infill is performed on each index plane separately. If two index planes are used, the indexes are stored interleaved: the first index belongs to the first index plane, the second index belongs to the second index plane, the third index belongs to the first index plane, and so on.

Index Infill Procedure

In ASTC, each block has an N*M*Q ordered grid of indices. N, M and Q may or may not match the dimensions of the actual block (e.g. it is possible to encode a 5x3 grid for an 8x8 block); if they don't match, then the grid is scaled so that its corner indexes align with the corner texels of the block, a bilinear index infill procedure is defined to interpolate an index for each texel. This procedure picks 1 to 4 indexes, and assigns each of them a weight; these weights are always a multiple of 1/16. The exact details of this interpolation procedure are specified below.

Color Endpoint Mode

In single-partition mode, the Color Endpoint Mode (CEM) field stores one of 16 possible values. Each of these specifies how many raw data values are encoded, and how to convert these raw values into two RGBA color endpoints. They can be summarized as follows:

List of Color Endpoint Modes

CEM	Description	Class	# of integers to represent each pair of color end points
0	LDR Luminance or Alpha, direct	0	2
1	LDR Luminance, base+offset	0	2
2	HDR Luminance, large range	0	2
3	HDR Luminance, small range	0	2
4	LDR Luminance+Alpha, direct	1	4
5	LDR Luminance+Alpha, base+offset	1	4
6	LDR RGB, base+scale	1	4
7	HDR RGB, base+scale	1	4
8	LDR RGB, direct	2	6
9	LDR RGB, base+offset	2	6
10	LDR RGB, base+scale plus two A	2	6
11	HDR RGB, direct	2	6
12	LDR RGBA, direct	3	8: D=0; 6: D=1
13	LDR RGBA, base+offset	3	8: D=0; 6: D=1
14	HDR RGB, direct + LDR Alpha	3	8: D=0; 6: D=1
15	HDR RGB, direct + HDR Alpha	3	8: D=0; 6: D=1

Description
LDR modes are supported in ASTC LDR profile.
HDR modes are only supported in ASTC HDR mode.

In 2-4 partition modes, the encoding of Color Endpoint Modes are listed in following tables, where the endpoint mode representation may take from 6 to 14 bits, of which the first 6 bits are stored just after the partition indices, and the remaining bits are stored just below the index bits at variable position in the remaining space.

Partition / Class Types		High bits				[1:0]
Same Class	**6b [5:0]**	[5:2] Color Endpoint Mode				0 0
Different Classes	**2-Partions 8b [7:0]**	[7:6] Mode in P1	[5:4] Mode in P0	[3:3] Class Select for P1	[2:2] Class Select for P0	0 1 (Class 0 & 1)
	3-Partions 11b [10:0]			[10:9] Mode in P2	[8:7] Mode in P1	1 0 (Class 1 & 2)
		[6:5] Mode in P0	[4:4] Class Select for P2	[3:3] Class Select for P1	[2:2] Class Select for P0	
	4-Partions 14b [13:0]	[13:12] Mode in P2	[11:10] Mode in P1	[9:8] Mode in P2	[7:6] Mode in P1	1 1 (Class 2 & 3)
		[5:5] Class Select for P3	[4:4] Class Select for P2	[3:3] Class Select for P1	[2:2] Class Select for P0	

More specifically, if the CEM selector value in bits [24:23] is not 00, then data layout is as follows:

List of Color Endpoint Class Types encoding under multi-partitions

Partitions							...	28	27	26	25	24	23
2	...	Index	M1				...	M0		C1	C0	CEM	
3	...	Index	M2	M1	M0		...	M0	C2	C1	C0	CEM	
4	...	Index	M3	M2	M1	M0	...	C3	C2	C1	C0	CEM	

In this view, each partition *i* has two fields. C*i* is the class selector bit, choosing between the two possible CEM classes (0 indicates the lower of the two classes), and M*i* is a two-bit field specifying the low bits of the color endpoint mode within that class. The additional bits appear at a variable bit position, immediately below the texel index data. The ranges used for the data values are not explicitly specified. Instead, they are derived from the number of available bits remaining after the configuration data and index data have been specified. Details of the decoding procedure for Color Endpoints can be found later.

Color Endpoint Data Size Determination

The size of the data used to represent color endpoints is not explicitly specified. Instead, it is determined from the index mode and number of partitions as follows:

```
config_bits = 17;
if (num_partitions>1)
if (single_CEM)
config_bits = 29;
else
config_bits = 24 + 3*num_partitions;

num_indices = M * N * Q; // size of index grid

if (dual_plane)
config_bits += 2;
num_indices *= 2;

index_bits = floor(num_indices*8*trits_in_index_range/5) +
floor(num_indices*7*quints_in_index_range/3) +
num_indices*bits_in_index_range;

remaining_bits = 128 - config_bits - index_bits;
num_CEM_pairs = base_CEM_class+1 + count_bits(extra_CEM_bits);
```

The CEM value range is then looked up from a table indexed by remaining bits and num_CEM_pairs. This table is initialized such that the range is as large as possible, consistent with the constraint that the number of bits required to encode num_CEM_pairs pairs of values is not more than the number of remaining bits. An equivalent iterative algorithm would be:

```
num_CEM_values = num_CEM_pairs*2;
for(range = each possible CEM range in descending order of size)
{
   CEM_bits = floor(num_CEM_values*8*trits_in_CEM_range/5) +
      floor(num_CEM_values*7*quints_in_CEM_range/3) +
      num_CEM_values*bits_in_CEM_range;
if (CEM_bits <= remaining_bits)
   break;
}
return range;
```

In cases where this procedure results in unallocated bits, these bits are not read by the decoding process and can have any value.

Void-Extent Blocks

As noted in the index mode, a specifically type of encoding is the void-extended type (2D), an efficient way to encode a constant color for large blocks of regions in texture. The data structure of a void extent is listed in following 2 tables as 2D and 3D blocks respectively.

Layout of 2D Void-Extend Block

127:112	111:96	95:80	79:64	63:51	50:38	37:25	24:12	11:10	9	8:0
A	B	G	R	T_high	T_low	S_high	S_low	Res:11	H	111111100

Layout of a 3D Void-Extent Block

127:112	111:96	95:80	79:64	63:55	54:46	45:37	36:28	27:19	18:10	9	8:0
A	B	G	R	R_high	R_low	T_high	T_low	S_high	S_low	H	111111100

Bit 9 H is the Dynamic Range flag, which indicates the format in which colors are stored. A 0 value indicates LDR, in which case the color components are stored as UNORM16 values. A 1 indicates HDR, in which case the color components are stored as FP16 values. If a void-extent block with HDR values is decoded in LDR mode, then the result will be the error color, opaque magenta, for all texels within the block. The low and height coordinate values are treated as unsigned integers and then normalized into the range 0..1 (by dividing by 2^{13}-1 for 2D or 2^{9}-1, for 3D respectively). The high values for each dimension must be greater than the corresponding low values, unless they are all all-1s. If all the coordinates are all-1s, then the void extent is ignored, and the block is simply a constant color block.The existence of single-color blocks with void extents must not produce results different from those obtained if these single-color blocks are defined without void-extents. Any situation in which the results would differ is invalid. Results from invalid void extents are undefined. If a void-extent appears in a MIPmap level other than the most detailed one, then the extent will apply to all of the more detailed levels too. This allows decoders to avoid sampling more detailed MIPmaps. If the more detailed MIPmap level is *not*

a constant color in this region, then the block may be marked as constant color, but without a void extent, as detailed above. If a void-extent extends to the edge of a texture, then filtered texture colors may not be the same color as that specified in the block, due to texture border colors, wrapping, or cube face wrapping. Care must be taken when updating or extracting partial image data that void-extents in the image do not become invalid.

Decoding Process

Content for this topic is currently under development.

Overview Decoding Flow

The goal for this feature is to reconstruct a cacheline (512b) of a target texture data at 4x4 region in UNORM8 A8R8G8B8 or 4x2 in FLT16 A16R16G16B16 with certain performance target, given the input texture coordinate (s,t,r). The scope of the u-architecture includes

- The additional surface format of the post decoding block, and the footprint (equivalent bpp). These are both global to each texture surface, and can be passed to the Sampler in the surface state via sampler messages.
- With post-scaled texture coordinate (u, v, p), the additional address calculation in FT to find the particular block location relative to the native block size specified in the surface state, as well as the relative texel position within that block. Assuming the block size for the block is Bu, Bv, Bp, the dimensions of a 2D surface as measured in block size tsize is:

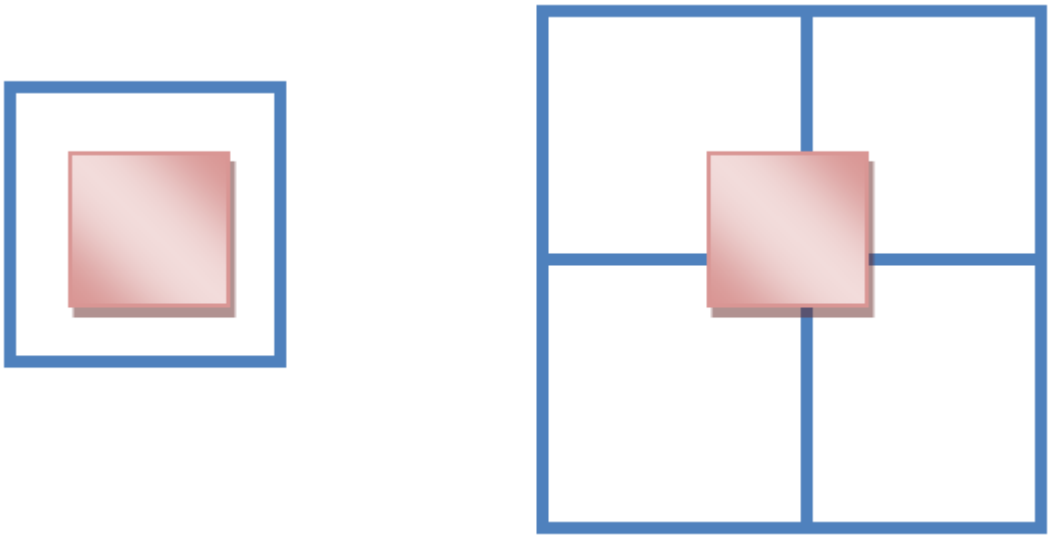
```
bw = MAX ( 2, (w+ tsize -1)/ tsize )
bh  = MAX (2, (h+ tsize -1)/ tsize)
```

Here the division is an integer division. The relationship between non-negative image coordinates [row,col] =[u, v] and block coordinates is

```
bu  = u / tsize  ;  buu = u % tsize;
bv  = v / tsize  ;  bvv =  v % tsize;
bp  = p / tsize  ;  bpp = p % tsize;
```

- With the selected sets of block size from 4x4 to 12x12 in 2D and 3x3x3 to 6x6x6 in 3D maps,1~4 blocks of source texture needs to be fetched, depending on whether the destination tile size (4x2 in FLT16 or 4x4 in UNORM8888) is inclusive or come across a few source blocks, as shown in Fig.

Destination tile is inclusive within one tile or across up to 4 tiles in source texture region



- Decode 1 to 4 128-bit ASTC compressed blocks fetched from DRAM in Sampler from ASTC compression format to either UNORM8 (LDR) or FLT16(HDR), reconstruct the texels needed in the texture filtering stage. The total decoding processing include:

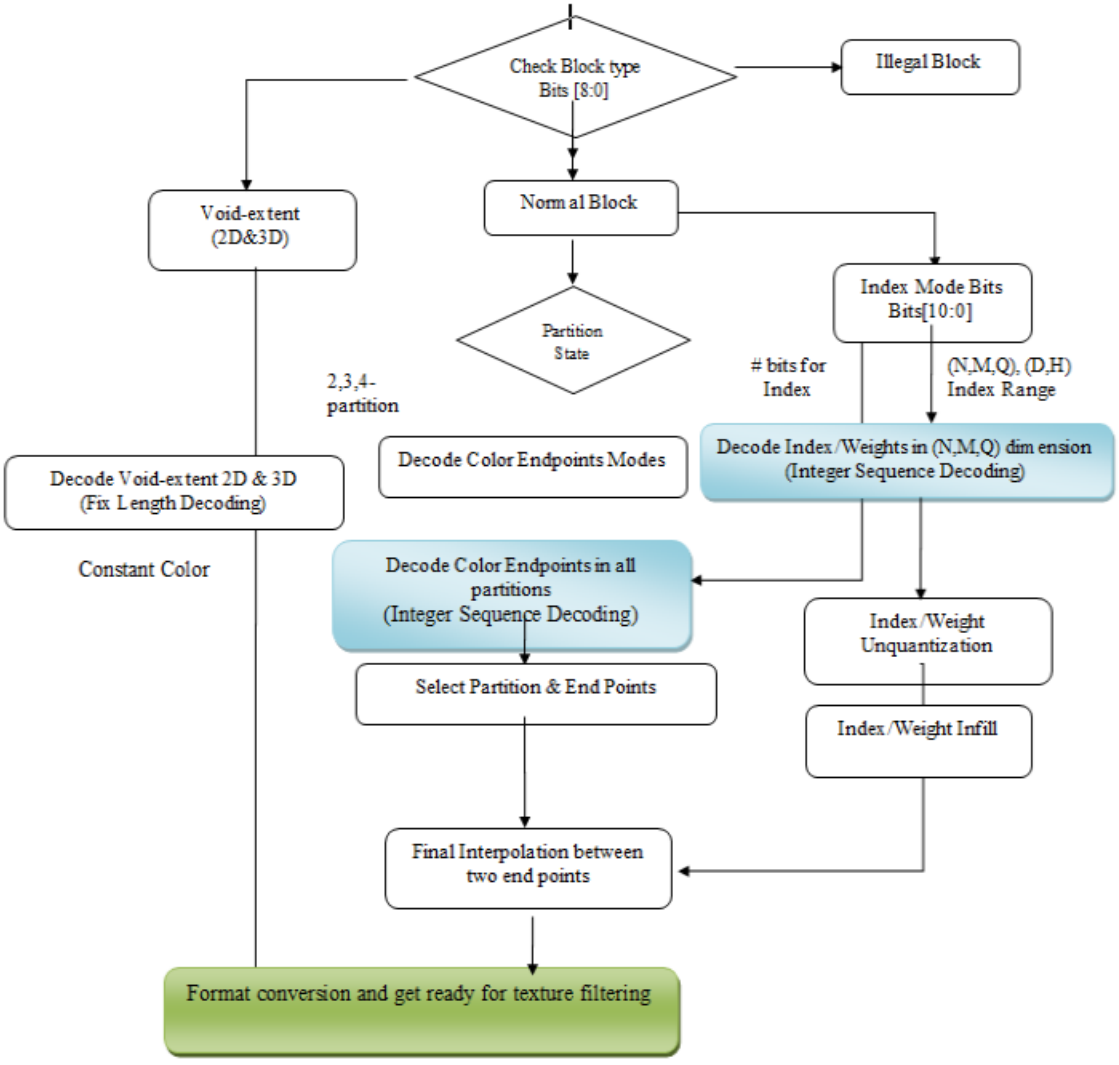
Front End Decoding Processing:

1. Detect if an ASTC block is a void-extent type, illegal type, or a normal non-void-extent type.
2. Decode the partition state – number of partitions in the current block.
3. Decode the index mode for the block include the partition seed and (N,M,Q) dimension of the compact sampling domain.
4. Decode the color endpoints modes in each partition.
5. Calculate the bit position and total # of bits used for Index.
6. Calculate the bit position and total # of bits and # of integers used in the Color endpoints in all partitions within the block.
7. With Integer Sequence Decoding, get all the indices in the compact domain defined by NxMxQ grid.
8. With Integer Sequence Decoding, get all the color end points from 16 modes in FLT16 for all partitions.

Back End Decoding Processing:

1. Reconstruct the indices at the selected sampling locations with infill scaling.
2. Find the partition from the partition seed at each sampling location.
3. Reconstruct the texture color value with the index and the pair of color end points at each sampling location.
4. If Block type is void extent, get the constant color from the high 64 bits and assign to the sampling location.
5. Convert the data to UNORM8 if LDR data is needed for the subsequent FL filtering process. Under void-extent block type,

Following is the flow diagram of the decoding process:



Integer Sequence Encoding

Both the index data and the endpoint color data are variable width, and are specified using a sequence of integer values. The range of each value in a sequence (e.g. a color index) is constrained. Since it is often the case that the most efficient range for these values is not a power of two, each value sequence is encoded using a technique known as "integer sequence encoding". This allows efficient, hardware-friendly packing and unpacking of values with non-power-of-two ranges. In a sequence, each value has an identical range. The range is specified in one of the following forms:

Value range	MSB encoding	LSB encoding	Value	Block	Packed block size
0 .. 2^n-1	-	*n* bit value *m* (*n* <= 8)	*m*	1	*n*
0 .. (3 * 2^n)-1	Base-3 "trit" value *t*	*n* bit value *m* (*n* <= 6)	*t* * 2^n + *m*	5	8 + 5*n
0 .. (5 * 2^n)-1	Base-5 "quint" value *q*	*n* bit value *m* (*n* <= 5)	*q* * 2^n + *m*	3	7 + 3*n

Since 3^5 is 243, it is possible to pack five trits into 8 bits (which has 256 possible values), so a trit can effectively be encoded as 1.6 bits. Similarly, since 5^3 is 125, it is possible to pack three quints into 7 bits (which has 128 possible values), so a quint can be encoded as 2.33 bits.

The encoding scheme packs the trits or quints, and then interleaves the *n* additional bits in positions that satisfy the requirements of an arbitrary length stream. This makes it possible to correctly specify lists of values whose length is not an integer multiple of 3 or 5 values. It also makes it possible to easily select a value at random within the stream. If there are insufficient bits in the stream to fill the final block, then unused (higher order) bits are assumed to be 0 when decoding.

To decode the bits for value number *i* in a sequence of bits *b*, both indexed from 0, perform the following:

If the range is encoded as *n* bits per value, then the value is bits *b*[*i***n*+*n*-1:*i***n*] – a simple multiplexing operation.

If the range is encoded using a trit, then each block contains 5 values (v0 to v4), each of which contains a trit (t0 to t4) and a corresponding LSB value (m0 to m4). The first bit of the packed block is bit floor(*i*/5)*(8+5**n*). The bits in the block are packed as follows (in this example, *n* is 4):

Trit-based Packing

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
T7			m4		T6		T5		m3		T4		m2		T3	T2		m1				T1	T0	m0				

The five trits t0 to t4 are obtained by bit manipulations of the 8 bits T[7:0] as follows:

```
if T[4:2] = 111
   C = { T[7:5], T[1:0] }; t4 = t3 = 2
else
   C = T[4:0]
   if T[6:5] = 11
      t4 = 2; t3 = T[7]
   else
      t4 = T[7]; t3 = T[6:5]
if C[1:0] = 11
   t2 = 2; t1 = C[4]; t0 = { C[3], C[2]&~C[3] }
else if C[3:2] = 11
   t2 = 2; t1 = 2; t0 = C[1:0]
else
   t2 = C[4]; t1 = C[3:2]; t0 = { C[1], C[0]&~C[1] }
```

Endpoint Unquantization

Each color endpoint is specified as a sequence of integers in a given range. These values are packed using integer sequence encoding, as a stream of bits stored from just above the configuration data, and growing upwards. Once unpacked, the values must be unquantized from their storage range, returning them to a standard range of 0..255. For bit-only representations, this is simple bit replication from the most significant bit of the value. For trit or quint-based representations, this involves a set of bit manipulations and adjustments to avoid the expense of full-width multipliers. This procedure ensures correct scaling, but scrambles the order of the decoded values relative to the encoded values. This must be compensated for using a table in the encoder.

The initial inputs to the procedure are denoted A, B, C and D and are decoded using the range as follows:

Range	Trits	Quints	Bits	Bit value	A (9 bits)	B (9 bits)	C (9 bits)	D (3 bits)
0..5	1		1	a	aaaaaaaaa	000000000	204	Trit value
0..9		1	1	a	aaaaaaaaa	000000000	113	Quint value
0..11	1		2	ba	aaaaaaaaa	b000b0bb0	93	Trit value

Range	Trits	Quints	Bits	Bit value	A (9 bits)	B (9 bits)	C (9 bits)	D (3 bits)
0..19		1	2	ba	aaaaaaaaa	b0000bb00	54	Quint value
0..23	1		3	cba	aaaaaaaaa	cb000cbcb	44	Trit value
0..39		1	3	cba	aaaaaaaaa	cb0000cbc	26	Quint value
0..47	1		4	dcba	aaaaaaaaa	dcb000dcb	22	Trit value
0..79		1	4	dcba	aaaaaaaaa	dcb0000dc	13	Quint value
0..95	1		5	edcba	aaaaaaaaa	edcb000ed	11	Trit value
0..159		1	5	edcba	aaaaaaaaa	edcb0000e	6	Quint value
0..191	1		6	fedcba	aaaaaaaaa	fedcb000f	5	Trit value

These are then processed as follows:

T= D * C + B;

T = T ^ A;

T = (A & 0x80) | (T » 2);

The multiply in the first line is nearly trivial as it only needs to multiply by 0, 1, 2, 3 or 4.

LDR Endpoint Decoding

The decoding method used depends on the Color Endpoint Mode (CEM) field, which specifies how many values are used to represent the endpoint. The CEM field also specifies how to take the *n* unquantized color endpoint values v0 to vn-1 and convert them into two RGBA color endpoints e0 and e1. The HDR Modes are more complex and do not fit neatly into the table. They are documented in following section. The LDR methods can be summarized as follows.

Color Endpoint Modes

CEM	Range	Description	# of end points	Endpoints Reconstruction
0	LDR	Luminance, direct	2	e0=(v0,v0,v0,0xFF); e1=(v1,v1,v1,0xFF);

1	LDR	Luminance,
		base+offset
			2	L0 = (v0»2)|(v1&0xC0); L1=L0+(v1&0x3F);
				if (L1>0xFF) { L1=0xFF; }
				e0=(L0,L0,L0,0xFF); e1=(L1,L1,L1,0xFF);

2	HDR	Luminance, large range	2	See next Section
3	HDR	Luminance, small range	2	See next Section
4	LDR	Luminance+Alpha, Direct	4	e0=(v0,v0,v0,v2); e1=(v1,v1,v1,v3);

5	LDR	Luminance+Alpha,
		base+offset
			4	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				e0=(v0,v0,v0,v2); e1=(v0+v1,v0+v1,v0+v1,v2+v3);
				clamp_unorm8(e0); clamp_unorm8(e1);

6	LDR	RGB, base+scale	4	e0=(v0*v3»8,v1*v3»8,v2*v3»8, 0xFF); e1=(v0,v1,v2,0xFF);
7	HDR	RGB, base+scale	4	See next Section

8	LDR	RGB,
		Direct
			6	s0= v0+v2+v4; s1= v1+v3+v5;
				if (s1>=s0){e0=(v0,v2,v4,0xFF); e1=(v1,v3,v5,0xFF); }
				else { e0=blue_contract(v1,v3,v5,0xFF);
				e1=blue_contract(v0,v2,v4,0xFF); }

9	LDR	RGB,
		base+offset
			6	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				bit_transfer_signed(v5,v4);
				if(v1+v3+v5 >= 0)
				{ e0=(v0,v2,v4,0xFF); e1=(v0+v1,v2+v3,v4+v5,0xFF);
				}

CEM	Range	Description	# of end points	Endpoints Reconstruction

				else
				{ e0=blue_contract(v0+v1,v2+v3,v4+v5,0xFF);
				e1=blue_contract(v0,v2,v4,0xFF); }
				clamp_unorm8(e0); clamp_unorm8(e1);

10	LDR	RGB, base+scale plus two A	6	e0=(v0*v3»8,v1*v3»8,v2*v3»8, v4); e1=(v0,v1,v2, v5);
11	HDR	RGB	6	See next Section

12	LDR	RGBA,
		direct
			8	s0= v0+v2+v4; s1= v1+v3+v5;
				if (s1>=s0){e0=(v0,v2,v4,v6); e1=(v1,v3,v5,v7); }
				else { e0=blue_contract(v1,v3,v5,v7);
				e1=blue_contract(v0,v2,v4,v6); }

13	LDR	RGBA,
		base+offset
			8	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				bit_transfer_signed(v5,v4);
				bit_transfer_signed(v7,v6);
				if(v1+v3+v5>=0) { e0=(v0,v2,v4,v6);
				e1=(v0+v1,v2+v3,v4+v5,v6+v7); }
				else { e0=blue_contract(v0+v1,v2+v3,v4+v5,v6+v7);
				e1=blue_contract(v0,v2,v4,v6); }
				clamp_unorm8(e0); clamp_unorm8(e1);

14	HDR	RGB + LDR Alpha	8	See next Section
15	HDR	RGB + HDR Alpha	8	See next Section

Mode 14 is special in that the alpha values are interpolated linearly, but the color components are interpolated logarithmically. This is the only endpoint format with mixed-mode operation, and will return the error value if encountered in LDR mode. The bit_transfer_signed procedure transfers a bit from one signed byte value (a) to another (b). The result is an 8-bit signed integer value and a 6-bit integer value sign extended to 8 bits. Note that, as is often the case, this is easier to express in hardware than in C:

bit_transfer_signed(uint16_t& a, uint16_t& b)

```
{
        b »= 1;
        b |= a & 0x80;
        a »= 1;
        a &= 0x3F;
        if( (a&0x20)!=0 ) a-=0x40;
}
```

For the purposes of this pseudocode, the signed bytes are passed in as unsigned 16-bit integers because the semantics of a right shift on a signed value in C are undefined.

The blue_contract procedure is used to give additional precision to RGB colors near grey:

```
color blue_contract( int r, int g, int b, int a )
{
        color c;
        c.r = (r+b) » 1;
        c.g = (g+b) » 1;
        c.b = b;
        c.a = a;
        return c;
}
```

The clamp_unorm8 procedure is used to clamp a color into the UNORM8 range:

```
void clamp_unorm8(color c)
        {
        if(c.r < 0) {c.r=0;} else if(c.r > 255) {c.r=255;}
        if(c.g < 0) {c.g=0;} else if(c.g > 255) {c.g=255;}
        if(c.b < 0) {c.b=0;} else if(c.b > 255) {c.b=255;}
        if(c.a < 0) {c.a=0;} else if(c.a > 255) {c.a=255;}
}
```

HDR Endpoint Decoding

The 6 HDR CEM modes on color endpoints reconstruction and surface formats are only used in full-profile ASTC texture in float 16 bit.

- HDR Endpoint Mode 2: HDR Luminance, large range

- HDR Endpoint Mode 3: HDR Luminance, small range
- HDR Endpoint Mode 7: HDR RGB, base + scale
- HDR Endpoint Mode 11: HDR RGB, direct
- HDR Endpoint Mode 14: HDR RGB, direct + LDR Alpha
- HDR Endpoint Mode 15: HDR RGB, direct + HDR Alpha

HDR Endpoint Mode 2 (HDR Luminance, Large Range)

Mode 2 represents luminance-only data with a large range. It encodes using two values (v0, v1). The complete decoding procedure is as follows:

```
If (v1 >= v0)
{
   y0 = (v0 « 4);
   y1 = (v1 « 4);
}
else {
   y0 = (v1 « 4) + 8;
   y1 = (v0 « 4) - 8;
}
// Construct RGBA result (0x780 is 1.0f)

e0 = (y0, y0, y0, 0x780);
e1 = (y1, y1, y1, 0x780);
```

HDR Endpoint Mode 3 (HDR Luminance, Small Range)

Mode 3 represents luminance-only data with a small range. It packs the bits for a base luminance value, together with an offset, into two values (v0, v1):

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	M	L[6:0]						
V1	X[3:0]				D[3:0]			

The bit field marked as X allocates different bits to L or d depending on the value of the mode bit M. The complete decoding procedure is as follows:

```
// Check mode bit and extract.
    If ((v0&0x80) !=0)
{
        y0 = ((v1 & 0xE0) « 4) | ((v0 & 0x7F) « 2);
        d = (v1 & 0x1F) « 2;
}
```

```
else {
        y0 = ((v1 & 0xF0) « 4) | ((v0 & 0x7F) « 1);
        d = (v1 & 0x0F) « 1;
}
// Add delta and clamp
y1 = y0 + d;
if(y1 > 0xFFF) { y1 = 0xFFF; }
// Construct RGBA result (0x780 is 1.0f)
e0 = (y0, y0, y0, 0x780);
e1 = (y1, y1, y1, 0x780);
```

HDR Endpoint Mode 7 (HDR RGB, Base+Scale)

Mode 7 packs the bits for a base RGB value, a scale factor, and some mode bits into the four values (v0, v1, v2, v3).

HDR Mode 7 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	M[3]	M[2]	R[5:0]					
V1	M[1]	X0	X1	G[4:0]				
V2	M[0]	X2	X3	B[4:0]				
V3	X4	X5	X6	S[4:0]				

The mode bits M[0:3] are a packed representation of an endpoint bit mode, together with the major component index. For modes 0 to 4, the component (red, green, or blue) with the largest magnitude is identified, and the values are swizzled to ensure that it is decoded from the red channel. The endpoint bit mode is used to determine the number of bits assigned to each component of the endpoint, and the destination of each of the extra bits X0 to X6, as follows:

Endpoint Bit Mode

Mode	Number of bits				Description of Extra Bits						
	R	G	B	Scale	X0	X1	X2	X3	X4	X5	X6
0	11	5	5	7	R[9]	R[8]	R[7]	R[10]	R[6]	S[6]	S[5]
1	11	6	6	5	R[8]	G[5]	R[7]	B[5]	R[6]	R[10]	R[9]
2	10	5	5	8	R[9]	R[8]	R[7]	R[6]	S[7]	S[6]	S[5]

Mode	Number of bits				Description of Extra Bits						
3	9	6	6	7	R[8]	G[5]	R[7]	B[5]	R[6]	S[6]	S[5]
4	8	7	7	6	G[6]	G[5]	B[6]	B[5]	R[6]	R[7]	S[5]
5	7	7	7	7	G[6]	G[5]	B[6]	B[5]	R[6]	S[6]	S[5]

The complete decoding procedure is as follows:

```
// Extract mode bits and unpack to major component and mode.
int modeval = ((v0 & 0xC0) » 6) | ((v1 & 0x80) » 5) | ((v2 & 0x80) » 4);
int majcomp;
int mode;
if( (modeval & 0xC ) != 0xC ) { majcomp = modeval » 2; mode = modeval & 3; }
else if( modeval != 0xF ) { majcomp = modeval & 3; mode = 4; }
else { majcomp = 0; mode = 5; }
// Extract low-order bits of r, g, b, and s.
int red = v0 & 0x3f;
int green = v1 & 0x1f;
int blue = v2 & 0x1f;
int scale = v3 & 0x1f;
// Extract high-order bits, which may be assigned depending on mode
int x0 = (v1 » 6) & 1; int x1 = (v1 » 5) & 1;
int x2 = (v2 » 6) & 1; int x3 = (v2 » 5) & 1;
int x4 = (v3 » 7) & 1; int x5 = (v3 » 6) & 1; int x6 = (v3 » 5) & 1;
// Now move the high-order xs into the right place.
int ohm = 1 « mode;
if( ohm & 0x30 ) green |= x0 « 6;
if( ohm & 0x3A ) green |= x1 « 5;
if( ohm & 0x30 ) blue |= x2 « 6;
if( ohm & 0x3A ) blue |= x3 « 5;
if( ohm & 0x3D ) scale |= x6 « 5;
if( ohm & 0x2D ) scale |= x5 « 6;
if( ohm & 0x04 ) scale |= x4 « 7;
if( ohm & 0x3B ) red |= x4 « 6;
```

```
if( ohm & 0x04 ) red |= x3 « 6;
if( ohm & 0x10 ) red |= x5 « 7;
if( ohm & 0x0F ) red |= x2 « 7;
if( ohm & 0x05 ) red |= x1 « 8;
if( ohm & 0x0A ) red |= x0 « 8;
if( ohm & 0x05 ) red |= x0 « 9;
if( ohm & 0x02 ) red |= x6 « 9;
if( ohm & 0x01 ) red |= x3 « 10;
if( ohm & 0x02 ) red |= x5 « 10;
// Shift the bits to the top of the 12-bit result.
static const int shamts[6] = { 1,1,2,3,4,5 };
int shamt = shamts[mode];
red «= shamt; green «= shamt; blue «= shamt; scale «= shamt;
// Minor components are stored as differences
if( mode != 5 ) { green = red - green; blue = red - blue; }
// Swizzle major component into place
if( majcomp == 1 ) swap( red, green );
if( majcomp == 2 ) swap( red, blue );
// Clamp output values, set alpha to 1.0
e1.r = clamp( red, 0, 0xFFF );
e1.g = clamp( green, 0, 0xFFF );
e1.b = clamp( blue, 0, 0xFFF );
e1.alpha = 0x780;
e0.r = clamp( red - scale, 0, 0xFFF );
e0.g = clamp( green - scale, 0, 0xFFF );
e0.b = clamp( blue - scale, 0, 0xFFF );
e0.alpha = 0x780;
```

HDR Endpoint Mode 11 (HDR RGB, Direct)

Mode 11 specifies two RGB values, which it calculates from a number of bitfields (a, b0, b1, c, d0 and d1) which are packed together with some mode bits into the six values (v0, v1, v2, v3, v4, v5):

HDR Mode 11 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	a[7:0]						
V1	m[0]	a[8]	c[5:0]				
V2	m[1]	X0	b0[5:0]				
V3	m[2]	X1	b1[5:0]				
V4	mj[0]	X2	X4	d0[4:0]			
V5	mj[1]	X3	X5	d1[4:0]			

If the major component bits mj[1:0] = b11, then the RGB values are specified directly as

HDR Mode 11 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	R0[11:4]						
V1	R1[11:4]						
V2	G0 [11:4]						
V3	G1[11:4]						
V4	1	B0[11:5]					
V5	1	B1[11:5]					

The mode bits m[2:0] specify the bit allocation for the different values, and the destinations of the extra bits X0 to X5:

Endpoint Bit Mode

Mode	Number of Bits				Description of Extra Bits					
	a	b	c	d	X0	X1	X2	X3	X4	X5
0	9	7	6	7	b0[6]	b1[6]	d0[6]	d1[6]	d0[5]	d1[5]
1	9	8	6	6	b0[6]	b1[6]	b0[7]	b1[7]	d0[5]	d1[5]
2	10	6	7	7	a[9]	c[6]	d0[6]	d1[6]	d0[5]	d1[5]
3	10	7	7	6	b0[6]	b1[6]	a[9]	c[6]	d0[5]	d1[5]

Mode	Number of Bits				Description of Extra Bits					
4	11	8	6	5	b0[6]	b1[6]	b0[7]	b1[7]	a[9]	a[10]
5	11	6	7	6	a[9]	a[10]	c[7]	c[6]	d0[5]	d1[5]
6	12	7	7	5	b0[6]	b1[6]	a[11]	c[6]	a[9]	a[10]
7	12	6	7	6	a[9]	a[10]	a[11]	c[6]	d0[5]	d1[5]

The complete decoding procedure is as follows:

```
// Find major component
int majcomp = ((v4 & 0x80) » 7) | ((v5 & 0x80) » 6);
// Deal with simple case first
if( majcomp == 3 )
{
e0 = (v0 « 4, v2 « 4, (v4 & 0x7f) « 5, 0x780);
e1 = (v1 « 4, v3 « 4, (v5 & 0x7f) « 5, 0x780);
return;
}
// Decode mode, parameters.
int mode = ((v1 & 0x80) » 7) | ((v2 & 0x80) » 6) | ((v3 & 0x80) » 5);
int va = v0 | ((v1 & 0x40) « 2);
int vb0 = v2 & 0x3f;
int vb1 = v3 & 0x3f;
int vc = v1 & 0x3f;
int vd0 = v4 & 0x7f;
int vd1 = v5 & 0x7f;
// Assign top bits of vd0, vd1.
static const int dbitstab[8] = {7,6,7,6,5,6,5,6};
vd0 = signextend( vd0, dbitstab[mode] );
vd1 = signextend( vd1, dbitstab[mode] );
// Extract and place extra bits
int x0 = (v2 » 6) & 1;
int x1 = (v3 » 6) & 1;
int x2 = (v4 » 6) & 1;
```

```
int x3 = (v5 » 6) & 1;
int x4 = (v4 » 5) & 1;
int x5 = (v5 » 5) & 1;
int ohm = 1 « mode;
if( ohm & 0xA4 ) va |= x0 « 9;
if( ohm & 0x08 ) va |= x2 « 9;
if( ohm & 0x50 ) va |= x4 « 9;
if( ohm & 0x50 ) va |= x5 « 10;
if( ohm & 0xA0 ) va |= x1 « 10;
if( ohm & 0xC0 ) va |= x2 « 11;
if( ohm & 0x04 ) vc |= x1 « 6;
if( ohm & 0xE8 ) vc |= x3 « 6;
if( ohm & 0x20 ) vc |= x2 « 7;
if( ohm & 0x5B ) vb0 |= x0 « 6;
if( ohm & 0x5B ) vb1 |= x1 « 6;
if( ohm & 0x12 ) vb0 |= x2 « 7;
if( ohm & 0x12 ) vb1 |= x3 « 7;
// Now shift up so that major component is at top of 12-bit value
int shamt = (modeval » 1) ^ 3;
va «= shamt; vb0 «= shamt; vb1 «= shamt;
vc «= shamt; vd0 «= shamt; vd1 «= shamt;
e1.r = clamp( va, 0, 0xFFF );
e1.g = clamp( va - vb0, 0, 0xFFF );
e1.b = clamp( va - vb1, 0, 0xFFF );
e1.alpha = 0x780;
e0.r = clamp( va - vc, 0, 0xFFF );
e0.g = clamp( va - vb0 - vc - vd0, 0, 0xFFF );
e0.b = clamp( va - vb1 - vc - vd1, 0, 0xFFF );
e0.alpha = 0x780;
if( majcomp == 1 )
{
```

```
swap( e0.r, e0.g ); swap( e1.r, e1.g );
}
else if( majcomp == 2 )
{
swap( e0.r, e0.b ); swap( e1.r, e1.b );
}
```

HDR Endpoint Mode 14 (HDR RGB, Direct + LDR Alpha)

Mode 14 specifies two RGBA values, using the eight values (v0, v1, v2, v3, v4, v5, v6, v7). First, the RGB values are decoded from (v0..v5) using the method from Mode 11. Then the alpha values are filled in from v6 and v7:

```
// Decode RGB as for mode 11
(e0,e1) = decode_mode_11(v0,v1,v2,v3,v4,v5)
// Now fill in the alphas
e0.alpha = v6;
e1.alpha = v7;
```

HDR Endpoint Mode 15 (HDR RGB, Direct + HDR Alpha)

Mode 15 specifies two RGBA values, using the eight values (v0, v1, v2, v3, v4, v5, v6, v7). First, the RGB values are decoded from (v0..v5) using the method from Mode 11. The alpha values are stored in values v6 and v7 as a mode and two values which are interpreted according to the mode:

HDR Mode 15 Alpha Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V6	M0	A[6:0]						
V7	M1	B[6:0]						

The alpha values are decoded from v6 and v7 as follows:

```
// Decode RGB as for mode 11
(e0,e1) = decode_mode_11(v0,v1,v2,v3,v4,v5)
// Extract mode bits
mode = ((v6 » 7) & 1) | ((v7 » 6) & 2);
v6 &= 0x7F;
v7 &= 0x7F;
```

```
if(mode==3)
{
// Directly specify alphas
        e0.alpha = v6 « 5;
        e1.alpha = v7 « 5;
}
else
{
        // Transfer bits from v7 to v6 and sign extend v7.
        v6 |= (v7 « (mode+1))) & 0x780;
        v7 &= (0x3F » mode);
        v7 ^= 0x20 » mode;
        v7 -= 0x20 » mode;
        v6 «= (4-mode);
        v7 «= (4-mode);
        // Add delta and clamp
        v7 += v6;
        v7 = clamp(v7, 0, 0xFFF);
        e0.alpha = v6;
        e1.alpha = v7;
}
```

Restrictions on Number of Partitions Per Block

Following table gives total number of partitions for each CEM mode given the restriction of total up to 16 integer values being decoded from the Integer Sequence Coding sequence.

Groups	Max Number of Partition	CEM Modes
(v0,v1)	4	0,1,2,3
(v0,v1,v2,v3)	4	4,5,6,7
(v0,v1,v2,v3,v4,v5)	3	8,9,10,11
(v0,v1,v2,v3,v4,v5,v6,v7)	2	12,13,14,15

Index Decoding

The index information is stored as a stream of bits, growing downwards from the most significant bit in the block. Bit *n* in the stream is thus bit 127-*n* in the block.

For each location in the index grid, a value (in the specified range) is packed into the stream. These are ordered in a raster pattern starting from location (0,0,0), with the X dimension increasing fastest, and the Z dimension increasing slowest. If dual-plane mode is selected, both indices are emitted together for each location, plane 0 first, then plane 1.

Index Unquantization

Each index plane is specified as a sequence of integers in a given range. These values are packed using integer sequence encoding.

Once unpacked, the values must be unquantized from their storage range, returning them to a standard range of 0..64. The procedure for doing so is similar to the color endpoint unquantization.

First, we unquantize the actual stored index values to the range 0..63.

For bit-only representations, this is simple bit replication from the most significant bit of the value.

For trit or quint-based representations, this involves a set of bit manipulations and adjustments to avoid the expense of full-width multipliers.

For representations with no additional bits, the results are as follows:

Index Unquantization Values

Range	0	1	2	3	4
0..2	0	32	63	-	-
0..4	0	16	32	47	63

For other values, we calculate the initial inputs to a bit manipulation procedure. These are denoted A, B, C and D and are decoded using the range as follows:

Index Unquantization Parameters

Range	Trits	Quints	Bits	Bit value	A (7 bits)	B (7 bits)	C (7 bits)	D (3 bits)
0..5	1		1	a	aaaaaaa	0000000	50	Trit
0..9		1	1	a	aaaaaaa	0000000	28	Quint
0..11	1		2	ba	aaaaaaa	b000b0b	23	Trit
0..19		1	2	ba	aaaaaaa	b0000b0	13	Quint

Range	Trits	Quints	Bits	Bit value	A (7 bits)	B (7 bits)	C (7 bits)	D (3 bits)
0..23	1		3	cba	aaaaaaa	cb000cb	11	Trit

These are then processed as follows:

T = D * C + B;

T = T ^ A;

T = (A & 0x20) | (T » 2);

The multiply in the first line is nearly trivial as it only needs to multiply by 0, 1, 2, 3 or 4. As a final step, for all types of value, the range is expanded from 0..63 up to 0..64 as follows:

if (T > 32) { T += 1; }

This allows the implementation to use 64 as a divisor during interpolation, which is much easier than using 63.

Infill Process

After unquantization, the indexes are subject to index selection and infill. The infill method is used to calculate the index for a texel position, based on the indices in the stored index grid array (which may be a different size). The procedure below must be followed exactly, to ensure bit exact results. The block size is specified as three dimensions along the s, t and r axes (Bs, Bt, Br). Texel coordinates within the block (s,t,r) can have values from 0 to one less than the block dimension in that axis.

For each block dimension, we compute scale factors (Ds, Dt, Dr)

Ds = floor((1024 + floor(Bs/2)) / (Bs-1));

Dt = floor((1024 + floor(Bt/2)) / (Bt-1));

Dr = floor((1024 + floor(Br/2)) / (Br-1));

Since the block dimensions are constrained, these are easily looked up in a table. These scale factors are then used to scale the (s,t,r) coordinates to a homogeneous coordinate (cs, ct, cr):

cs = Ds * s;

ct = Dt * t;

cr = Dr * r;

This homogeneous coordinate (cs, ct, cr) is then scaled again to give a coordinate (gs, gt, gr) in the index-grid space . The index-grid is of size (N, M, Q), as specified in the index mode field:

gs = (cs*(N-1)+32) » 6;

gt = (ct*(M-1)+32) » 6;

gr = (cr*(Q-1)+32) » 6;

The resulting coordinates may be in the range 0..176. These are interpreted as 4:4 unsigned fixed point numbers in the range 0.0 .. 11.0. If we label the integral parts of these (js, jt, jr) and the fractional parts (fs, ft, fr), then:

js = gs » 4; fs = gs & 0x0F;

jt = gt » 4; ft = gt & 0x0F;

jr = gr » 4; fr = gr & 0x0F;

These values are then used to interpolate between the stored indices. This process differs for 2D and 3D.

For 2D, bilinear interpolation is used:

v0 = js + jt*N;

p00 = decode_index(v0);

p01 = decode_index(v0 + 1);

p10 = decode_index(v0 + N);

p11 = decode_index(v0 + N + 1);

The function decode_index(n) decodes the nth index in the stored index stream. The values p00 to p11 are the indices at the corner of the square in which the texel position resides. These are then weighted using the fractional position to produce the effective index i as follows:

w11 = (fs*ft+8) » 4;

w10 = ft – w11;

w01 = fs – w11;

w00 = 16 – fs – ft + w11;

i = (p00*w00 + p01*w01 + p10*w10 + p11*w11 + 8) » 4;

For 3D, simplex interpolation is used as it is cheaper than a naïve trilinear interpolation. First, we pick some parameters for the interpolation based on comparisons of the fractional parts of the texel position:

fs>ft	ft>fr	fs>fr	s1	s2	w0	w1	w2	w3
True	True	True	1	N	16-fs	fs-ft	ft-fr	fr
False	True	True	N	1	16-ft	ft-fs	fs-fr	fr
True	False	True	1	N*M	16-fs	fs-fr	fr-ft	ft
True	False	False	N*M	1	16-fr	fr-fs	fs-ft	ft
False	True	False	N	N*M	16-ft	ft-fr	fr-fs	fs
False	False	False	N*M	N	16-fr	fr-ft	ft-fs	fs

The effective index *i* is then calculated as:

v0 = js + jt*N + jr*N*M;

p0 = decode_index(v0);

p1 = decode_index(v0 + s1);

p2 = decode_index(v0 + s1 + s2);

p3 = decode_index(v0 + N*M + N + 1);

i = (p0*w0 + p1*w1 + p2*w2 + p3*w3 + 8) » 4;

Index Application

Once the effective index *i* for the texel has been calculated, the color endpoints are interpolated and expanded. For LDR endpoint modes, each color component C is calculated from the corresponding 8-bit endpoint components C0 and C1 as follows:

If sRGB conversion is not enabled, C0 and C1 are first expanded to 16 bits by bit replication:

C0 = (C0 « 8) | C0; C1 = (C1 « 8) | C1;

If sRGB conversion is enabled, C0 and C1 are expanded to 16 bits differently, as follows:

C0 = (C0 « 8) | 0x80; C1 = (C1 « 8) | 0x80;

C0 and C1 are then interpolated to produce a UNORM16 result C:

C = floor((C0*(64-i) + C1*i + 32)/64)

If sRGB conversion is enabled, the top 8 bits of the interpolation result are passed to the external sRGB conversion block. Otherwise, if C = 65535, then the final result is 1.0 (0x3C00) otherwise C is divided by 65536 and the infinite-precision result of the division is converted to FP16 with round-to-zero semantics. For HDR endpoint modes, color values are represented in a 12-bit logarithmic representation, and interpolation occurs in a piecewise-approximate logarithmic manner as follows:

In LDR mode, the error result is returned.

In HDR mode, the color components from each endpoint, C0 and C1, are initially shifted left 4 bits to become 16-bit integer values and these are interpolated in the same way as LDR. The 16-bit value C is then decomposed into the top five bits, E, and the bottom 11 bits M, which are then processed and recombined with E to form the final value Cf:

C = floor((C0*(64-i) + C1*i + 32)/64)

E = (C&0xF800) » 11; M = C&0x7FF;

if (M < 512) { Mt = 3*M; }

else if (M >= 1536) { Mt = 5*M – 2048; }

else { Mt = 4*M – 512; }

Cf = (E«10) + (Mt»3)

This final value Cf is interpreted as an IEEE FP16 value. If the result is +Inf or NaN, it is converted to the bit pattern 0x7BFF, which is the largest representable finite value.

Dual-Plane Decoding

If dual-plane mode is disabled, all of the endpoint components are interpolated using the same index value. If dual-plane mode is enabled, two indices are stored with each texel. One component is then selected to use the second index for interpolation, instead of the first index. The first index is then used for all other components.

The component to treat specially is indicated using the 2-bit Color Component Selector (CCS) field as follows:

Dual Plane Color Component Selector Values

Value	Index 0	Index 1
0	GBA	R
1	RBA	G
2	RGA	B
3	RGB	A

The CCS bits are stored at a variable position directly below the index bits and any additional CEM bits.

Partition Pattern Generation

When multiple partitions are active, each texel position is assigned a partition index. This partition index is calculated using a seed (the partition pattern index), the texel's x,y,z position within the block, and the number of partitions. An additional argument, small_block, is set to 1 if the number of texels in the block is less than 31, otherwise it is set to 0. The full partition selection algorithm is as follows:

```
int select_partition(int seed, int x, int y, int z,
int partitioncount, int small_block)
{
    if( small_block ){ x «= 1; y «= 1; z «= 1; }
    seed += (partitioncount-1) * 1024;
    uint32_t rnum = hash52(seed);
    uint8_t seed1 = rnum & 0xF;
    uint8_t seed2 = (rnum » 4) & 0xF;
    uint8_t seed3 = (rnum » 8) & 0xF;
    uint8_t seed4 = (rnum » 12) & 0xF;
    uint8_t seed5 = (rnum » 16) & 0xF;
```

```
    uint8_t seed6 = (rnum » 20) & 0xF;
    uint8_t seed7 = (rnum » 24) & 0xF;
    uint8_t seed8 = (rnum » 28) & 0xF;
    uint8_t seed9 = (rnum » 18) & 0xF;
    uint8_t seed10 = (rnum » 22) & 0xF;
    uint8_t seed11 = (rnum » 26) & 0xF;
    uint8_t seed12 = ((rnum » 30) | (rnum « 2)) & 0xF;
    seed1 *= seed1; seed2 *= seed2; seed3 *= seed3; seed4 *= seed4;
    seed5 *= seed5; seed6 *= seed6; seed7 *= seed7; seed8 *= seed8;
    seed9 *= seed9; seed10 *= seed10; seed11 *= seed11; seed12 *= seed12;
    int sh1, sh2, sh3;
    if( seed & 1 )
    { sh1 = (seed & 2 ? 4 : 5); sh2 = (partitioncount == 3 ? 6 : 5); }
    else
    { sh1 = (partitioncount == 3 ? 6 : 5); sh2 = (seed & 2 ? 4 : 5); }
    sh3 = (seed & 0x10) ? sh1 : sh2:
    seed1 »= sh1; seed2 »= sh2; seed3 »= sh1; seed4 »= sh2;
    seed5 »= sh1; seed6 »= sh2; seed7 »= sh1; seed8 »= sh2;
    seed9 »= sh3; seed10 »= sh3; seed11 »= sh3; seed12 »= sh3;
    int a = seed1*x + seed2*y + seed11*z + (rnum » 14);
    int b = seed3*x + seed4*y + seed12*z + (rnum » 10);
    int c = seed5*x + seed6*y + seed9 *z + (rnum » 6);
    int d = seed7*x + seed8*y + seed10*z + (rnum » 2);
    a &= 0x3F; b &= 0x3F; c &= 0x3F; d &= 0x3F;
    if( partitioncount < 4 ) d = 0;
    if( partitioncount < 3 ) c = 0;
    if( a >= b && a >= c && a >= d ) return 0;
    else if( b >= c && b >= d ) return 1;
    else if( c >= d ) return 2;
    else return 3;
}
```

As has been observed before, the bit selections are much easier to express in hardware than in C.

The seed is expanded using a hash function hash52, which is defined as follows:

```
uint32_t hash52( uint32_t p )
{
        p ^= p » 15; p -= p « 17; p += p « 7; p += p « 4; p ^= p » 5;
        p += p « 16; p ^= p » 7; p ^= p » 3; p ^= p « 6; p ^= p » 17;
        return p;
}
```

This assumes that all operations act on 32-bit values

Data Size Determination

The size of the data used to represent color endpoints is not explicitly specified. Instead, it is determined from the index mode and number of partitions as follows:

```
config_bits = 17;
if(num_partitions>1)
if(single_CEM)
        config_bits = 29;
else
        config_bits = 24 + 3*num_partitions;
num_indices = M * N * Q; // size of index grid
if(dual_plane)
config_bits += 2;
num_indices *= 2;
index_bits = ceil(num_indices*8*trits_in_index_range/5) +
        ceil(num_indices*7*quints_in_index_range/3) +
                num_indices*bits_in_index_range;
remaining_bits = 128 – config_bits – index_bits;
num_CEM_pairs = base_CEM_class+1 + count_bits(extra_CEM_bits);
```

The CEM value range is then looked up from a table indexed by remaining bits and num_CEM_pairs. This table is initialized such that the range is as large as possible, consistent with the constraint that the number of bits required to encode num_CEM_pairs pairs of values is not more than the number of remaining bits.

An equivalent iterative algorithm would be:

```
num_CEM_values = num_CEM_pairs*2;
for(range = each possible CEM range in descending order of size)
{
        CEM_bits = ceil(num_CEM_values*8*trits_in_CEM_range/5) +
        ceil(num_CEM_values*7*quints_in_CEM_range/3) +
        num_CEM_values*bits_in_CEM_range;
        if(CEM_bits <= remaining_bits)
        break;
}
return range;
```

In cases where this procedure results in unallocated bits, these bits are not read by the decoding process and can have any value.

3D Void-Extent Blocks

The layout of a 3D Void-Extent block is as follows:

127:112	111:96	95:80	79:64	63:55	54:46	45:37	36:28	27:19	18:10	9:9	8:0
A	B	G	R	P_high	P_low	T_high	T_low	S_high	S_low	D	111111100

Bit 9 is the Dynamic Range flag, which indicates the format in which colors are stored. Value 0 indicates LDR, in which case the color components are stored as UNORM16 values, while value 1 indicates HDR, in which case the color components are stored as FP16 values.

The reason for the storage of UNORM16 values in the LDR case is due to the possibility that the value will need to be passed on to sRGB conversion. By storing the color value in the format which comes out of the interpolator, before the conversion to FP16, we avoid having to have separate versions for sRGB and linear modes.

If a void-extent block with HDR values is decoded in LDR mode, then the result will be the error color, opaque magenta, for all texels within the block.

The minimum and maximum coordinate values are treated as unsigned integers and then normalized into the range 0..1 (by dividing by 2^{13}-1 or 2^{9}-1, for 2D and 3D respectively). The maximum values for each dimension must be greater than the corresponding minimum values, unless they are all all-1s. If all the coordinates are all-1s, then the void extent is ignored, and the block is simply a constant-color block.

Illegal Encodings

In ASTC, there is a variety of ways to encode an illegal block. Decoders are required to recognize all illegal blocks and emit the standard Error Block color value upon encountering an illegal block. The standard Error Block color value is opaque magenta (R, G, B, A) = (0xFF, 0x00, 0xFF, 0xFF) in the LDR operation mode, and a vector of NaNs (**R, G, B, A**)=(NaN, NaN, NaN, NaN) in the HDR operation mode. It is recommended that the NaN be encoded as the bit-pattern 0xFFFF.

Here is a comprehensive list of situations that represent illegal block encodings:

- The index bit mode specified is one of the modes explicitly listed as Reserved.
- An index bit mode has been specified that would require more than 64 indexes total.
- An index bit mode has been specified that would require more than 96 bits for the Index Integer Sequence Encoding.
- An index bit mode has been specified that would require fewer than 24 bits for the Index Integer Sequence Encoding.
- The size of the index grid exceeds the size of the block footprint in any dimension.
- Color endpoint modes have been specified such that the Color Integer Sequence Encoding would require more than 18 integers.
- The number of bits available for color endpoint encoding after all the other fields have been counted is less than *ceil(13C/5)* where C is the number of color endpoint integers (this would restrict color integers to a range smaller than 0..5, which is not supported).
- Dual Index Mode is enabled for a block with 4 partitions.
- Void-Extent blocks where the low coordinate for some texture axis is greater than or equal to the high coordinate.
- Under 3D mode, the depth (Q) is not 1

In LDR mode, a block which has both HDR and LDR endpoint modes assigned to different partitions is not an error block. Only those texels which belong to the HDR partition will result in the error color. Texels belonging to a LDR partition will be decoded as normal.

Profile Support

In order to ease verification and accelerate adoption, an LDR-only subset of the full ASTC specification has been made available.

Implementations of this LDR Profile must satisfy the following requirements:

- All textures with valid encodings for LDR Profile must decode identically using either a LDR Profile or Full Profile decoder.
- All features included only in the Full Profile must be treated as reserved in the LDR Profile, and return the error color on decoding.
- Any sequence of API calls valid for the LDR Profile must also be valid for the Full Profile and return identical results when given a texture encoded for the LDR Profile.

The feature subset for the LDR profile is:

- 2D textures only.
- Only those block sizes listed in Table 5 are supported.
- LDR operation mode only.
- Only LDR endpoint formats must be supported namely formats 0, 1, 4, 5, 6, 8, 9, 10, 12, 13.
- Decoding from a HDR endpoint results in the error color.
- Interpolation returns UNORM8 results when used in conjunction with sRGB.
- LDR void extent blocks must be supported, but void extents may not be checked.

Video Pixel/Texel Formats

This section describes the “video” pixel/texel formats with respect to memory layout. See the Overlay chapter for a description of how the Y, U, V components are sampled.

Packed Memory Organization

Color components are all 8 bits in size for YUV formats. For YUV 4:2:2 formats each DWord will contain two pixels and only the byte order affects the memory organization.

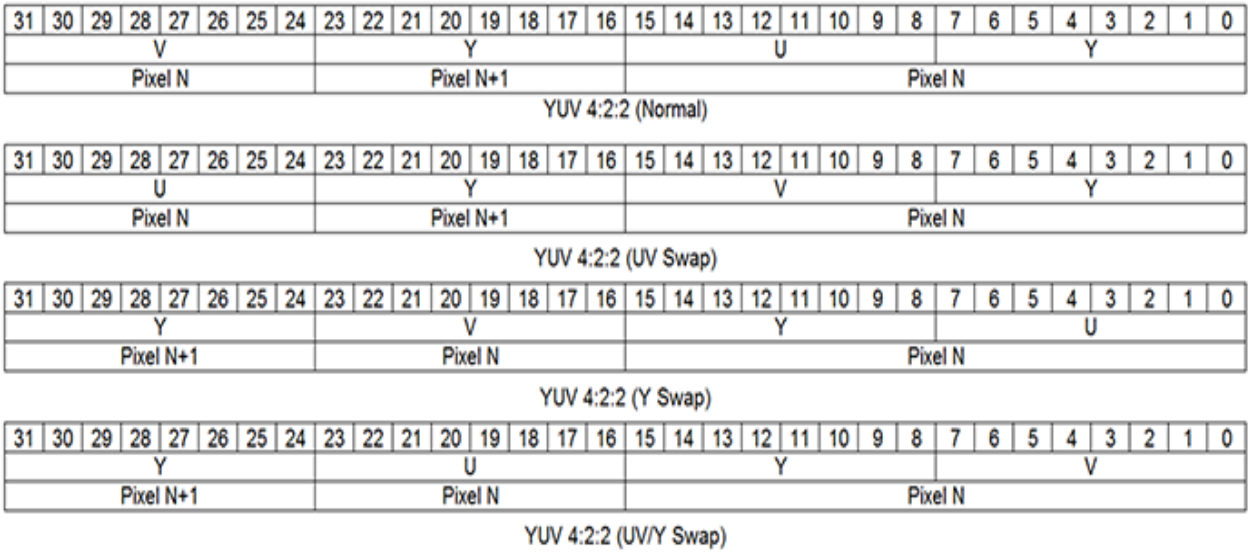
The following four YUV 4:2:2 surface formats are supported, listed with alternate names:

- YCRCB_NORMAL (YUYV/YUY2)
- YCRCB_SWAPUVY (VYUY) (R8G8_B8G8_UNORM)
- YCRCB_SWAPUV(YVYU) (G8R8_G8B8_UNORM)
- YCRCB_SWAPY (UYVY)

The channels are mapped as follows:

Cr (V)	Red
Y	Green
Cb (U)	Blue

Memory layout of packed YUV 4:2:2 formats



Planar Memory Organization

Planar formats use what could be thought of as separate buffers for the three color components. Because there is a separate stride for the Y and U/V data buffers, several memory footprints can be supported.

The 3D sampler supports direct sampling and filtering of planar video surfaces such as YV12 and NV12.

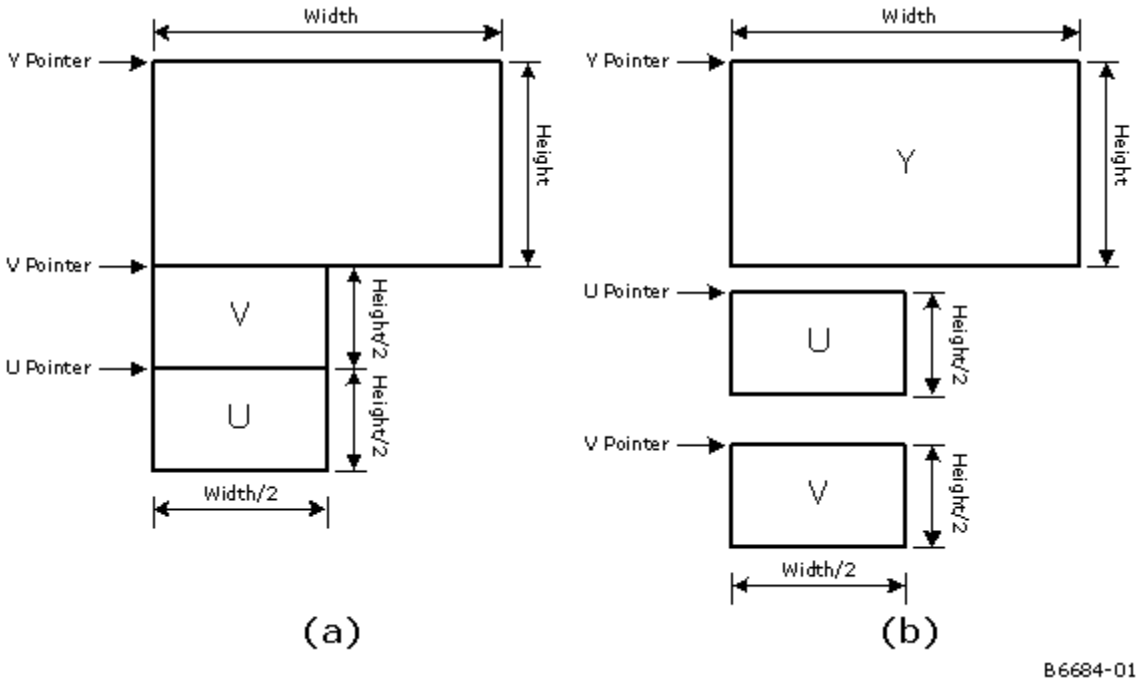
Programming Note	
Context:	Tiling of Planar Surface
Tiling of planar surfaces (tileX, tileY, tileYf, or tileYs) is only supported for planar surfaces where the chroma plane is full-pitch (e.g. NV21). In this case, the field **Y Offset for U or UV Plane** in the RENDER_SURFACE_STATE must be programmed to force the UV plane to be at the start of a tile.	

Programming Note	
Context:	NV21 Support
Sampling of NV21 surface format is supported by swapping the U and V channels when sampling the surface. This can be done by programming the **Shader Channel Select** in the RENDER_SURFACE_STATE for the Red and Blue Channels.	

The following figure shows two types of memory organization for the YUV 4:2:0 planar video data:

1. The memory organization of the common YV12 data, where all three planes are contiguous and the strides of U and V components are half of that of the Y component.
2. An alternative memory structure that the addresses of the three planes are independent but satisfy certain alignment restrictions.

YUV 4:2:0 Format Memory Organization

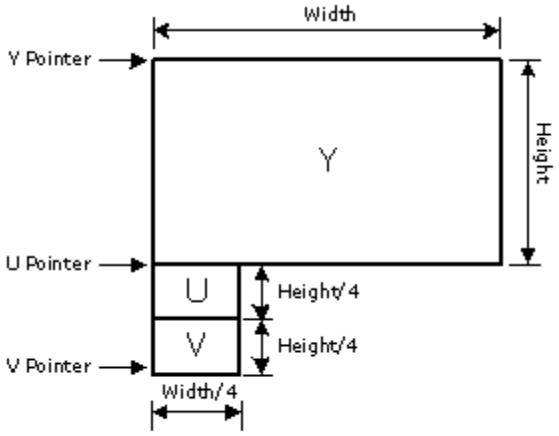


The following figure shows memory organization of the planar YUV 4:1:0 format where the planes are contiguous.

Note: The chroma planes (U and V), when separate (case b above) are treated as half-pitch with respect to the Y plane.

Workaround
When using Planar formats for YUV with half-pitch chroma planes (e.g. YV12), and fenced tiling is not supported
LINEAR filtering of Planar YUV surfaces such as YV12 using the 3D sampler is done after the U and V have been replicated to form a YUV444 texels. This means that the U and V components will effectively be point-sampled rather than filtered. Acheive true filtering of the U and V components, the 3 planes of the YUV surface must be bound as separate surfaces, and the filtering must be done on each individually.

YUV 4:1:0 Format Memory Organization



B6685-01

The table below shows how position within a Planar YUV surface chroma plane is calculated for various cases ot U and V pitch and position. It also shows restrictions on the alignment of the chroma planes in memory for non-interleaved (YV12) and interleaved chroma (e.g. NV12) is used.

Case	Interleave Chroma	Pitch	Vertical U/V Offset
YUV with Half Pitch Chroma	No	Half	When U is below Y Y_Uoffset = Y_Height * 2 Y_Voffset = Y_Height * 2 + V_Height When V is below Y Y_Uoffset = Y_Height * 2 + V_Height Y_Voffset = Y_Height * 2
YUV with Full Pitch Chroma	Yes	Full	When U is below Y Y_Uoffset = Y_Height Y_Voffset = Y_Height + V_Height When V is below Y Y_Uoffset = Y_Height + V_Height Y_Voffset = Y_Height
YUV for Media Sampling	Yes	Always Full	Same as 3D full pitch

Programming Note
Context: Planar YUV surfaces cannot be 1D surface types.
Because there is a requirement that the height of the Y plane of a planar surface must be a greater than 1, it cannot be programmed to be a **Surface Type** of SURFTYPE_1D.

Programming Note	
Context:	MIP Filtering
Surface state cannot have (**MIP Mode Filter** != NONE) for Planar YUV surfaces (e.g. PLANAR_420_8).	

Programming Note	
Context:	Standard Tiling
Planar YUV does not support MIP Tails as part of Standard Tiling. The MIP Tail Start field in RENDER_SURFACE_STATE must be programmed to 15.	

Raw Format

A format called "RAW" is available that is only supported with the untyped surface read/write, block, scattered, and atomic operation data port messages. It means that the surface has no inherent format. Surfaces of type RAW are addressed with byte-based offsets. The RAW surface format can be applied only to surface types of BUFFER and STRBUF.

Surface Memory Organizations

See *Memory Interface Functions* chapter for a discussion of tiled vs. linear surface formats.

Display, Overlay, Cursor Surfaces

These surfaces are memory image buffers (planes) used to refresh a display device in non-VGA mode. See the Display chapter for specifics on how these surfaces are defined/used.

2D Render Surfaces

These surfaces are used as general source and/or destination operands in 2D BLT operations.

Note that there is no coherency between 2D render surfaces and the texture cache. Software must explicitly invalidate the texture cache before using a texture that has been modified via the BLT engine.

See the 2D Instruction and 2D Rendering chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

2D Monochrome Source

These 1 BPP (bit per pixel) surfaces are used as source operands to certain 2D BLT operations, where the BLT engine expands the 1 BPP source to the required color depth.

The texture cache stores any monochrome sources. There is no mechanism to maintain coherency between 2D render surfaces and texture-cached monochrome sources. Software must explicitly invalidate the texture cache before using a memory-based monochrome source that has been modified via the BLT engine. (Here the assumption is that SW enforces memory-based monochrome source surfaces as read-only surfaces.)

See the 2D Instruction and 2D Rendering chapters for specifics on how these surfaces are used, restrictions on their size, placement, coherency rules, etc.

2D Color Pattern

Color pattern surfaces are used as special pattern operands in 2D BLT operations.

The device uses the texture cache to store color patterns. There is no mechanism to maintain coherency between 2D render surfaces and (texture)-cached color patterns. Software is required to explicitly invalidate the texture cache before using a memory-based color pattern that has been modified via the BLT engine. (Here the assumption is that SW enforces memory-based color pattern surfaces as read-only surfaces.)

See the *2D Instruction* and *2D Rendering* chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

3D Color Buffer (Destination) Surfaces

3D Color Buffer surfaces hold per-pixel color values for use in the 3D Pipeline. The 3D Pipeline always requires a Color Buffer to be defined.

See the Non-Video Pixel/Texel Formats section in this chapter for details on the Color Buffer pixel formats. See the 3D Instruction and 3D Rendering chapters for Color Buffer usage details.

The Color Buffer is defined as the BUFFERID_COLOR_BACK memory buffer via the 3DSTATE_BUFFER_INFO instruction. That buffer can be mapped to LM or SM (snooped or unsnooped), and can be linear or tiled. When both the Depth and Color Buffers are tiled, the respective Tile Walk directions must match.

When a linear Color Buffer and a linear Depth Buffer are used together:

- The buffers may have different pitches, though both pitches must be a multiple of 32 bytes.
- The buffers must be co-aligned with a 32-byte region.

3D Depth Buffer Surfaces

Depth Buffer surfaces hold per-pixel depth values and per-pixel stencil values for use in the 3D Pipeline. The 3D Pipeline does not require a Depth Buffer in general, though a Depth Buffer is required to perform non-trivial Depth Test and Stencil Test operations.

The Depth Buffer is specified via the 3DSTATE_DEPTH_BUFFER command. See the description of that instruction in *Windower* for restrictions.

See *Depth Buffer Formats* below for a summary of the possible depth buffer formats. See the Depth Buffer Formats section in this chapter for details on the pixel formats. See the *Windower* and *DataPort* chapters for details on the usage of the Depth Buffer.

Depth Buffer Formats

DepthBufferFormat / DepthComponent	BPP (Bits Per Pixel)	Description
D32_FLOAT_S8X24_UINT	64	32-bit floating point Z depth value in first DWord, 8-bit stencil in lower byte of second DWord
D32_FLOAT	32	32-bit floating point Z depth value
D24_UNORM_S8_UINT	32	24-bit fixed point Z depth value in lower 3 bytes, 8-bit stencil value in upper byte
D16_UNORM	16	16-bit fixed point Z depth value

3D Separate Stencil Buffer Surfaces

Separate Stencil Buffer surfaces hold per-pixel stencil values for use in the 3D Pipeline. Note that the 3D Pipeline does not require a Stencil Buffer to be allocated, though a Stencil Buffer is required to perform non-trivial Stencil Test operations.

UNRESOLVED CROSS-REFERENCE, Depth Buffer Formats summarizes Stencil Buffer formats. Refer to the Stencil Buffer Formats section in this chapter for details on the pixel formats. Refer to the *Windower* chapters for Stencil Buffer usage details.

The Stencil buffer is specified via the 3DSTATE_STENCIL_BUFFER command. See that instruction description in *Windower* for restrictions.

Depth Buffer Formats

DepthBufferFormat / DepthComponent	BPP (bits per pixel)	Description
R8_ UNIT	8	8-bit stencil value in a byte

Surface Layout and Tiling

This section explains how various surface types (1D, 2D, 3D, and Cube) are laid out in memory. Most of the information in this section is independent of tiling. The concept of tiling can be laid on top of information. Wherever there is a specific difference it will be called out.

For Tiling (TileY, TileYs etc.), see the Address Tiling Function Introduction section which provides detailed information on how tiles are organized and laid out.

Maximum Surface Size in Bytes

In addition to restrictions on maximum height, width, and depth, surfaces are also restricted to a maximum size of 2^38 bytes. All pixels within the surface must be contained within 2^38 bytes of the base address.

Tiling

To improve efficiency in memory accesses, most surfaces can be laid out using a tiling scheme.

Supported Legacy Tiling Modes:

- **TileY**
- **TileX**
- **TileW**

Supported Tiled Resource Modes

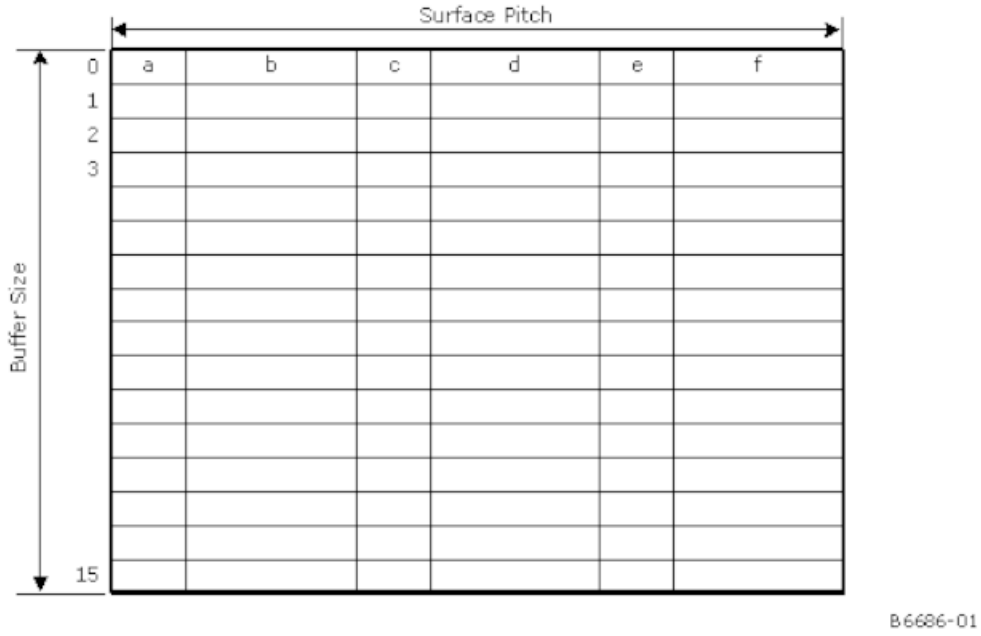
- **TileYF:** 4KB tiling mode based on TileY
- **TileYS:** 64KB tiling mode based on TileY

These modes are described in the Address Tiling Function Introduction volume.

Typed Buffers

A typed buffer is an array of structures. Each structure contains up to 2048 bytes of elements. Each element is a single surface format using one of the supported surface formats depending on how the surface is being accessed. The surface pitch state for the surface specifies the size of each structure in bytes.

The buffer is stored in memory contiguously with each element in the structure packed together, and the first element in the next structure immediately following the last element of the previous structure. Buffers are supported only in linear memory.



Typed buffers are accessed using a surface state for each structure element (a,b,c, etc. in the diagram above). The surface state for element “b” (for example) contains the surface format of element “b” (which may differ from other elements), the base address points to element “b” in the first structure (slice 0 of the array). The pitch for all of the elements in the buffer is the same value, and the surface type of each element is SURFTYPE_BUFFER.

The offset into the typed buffer is given by the following equation:

```
Offset = (V * Pitch) + U
```

MIP Layout

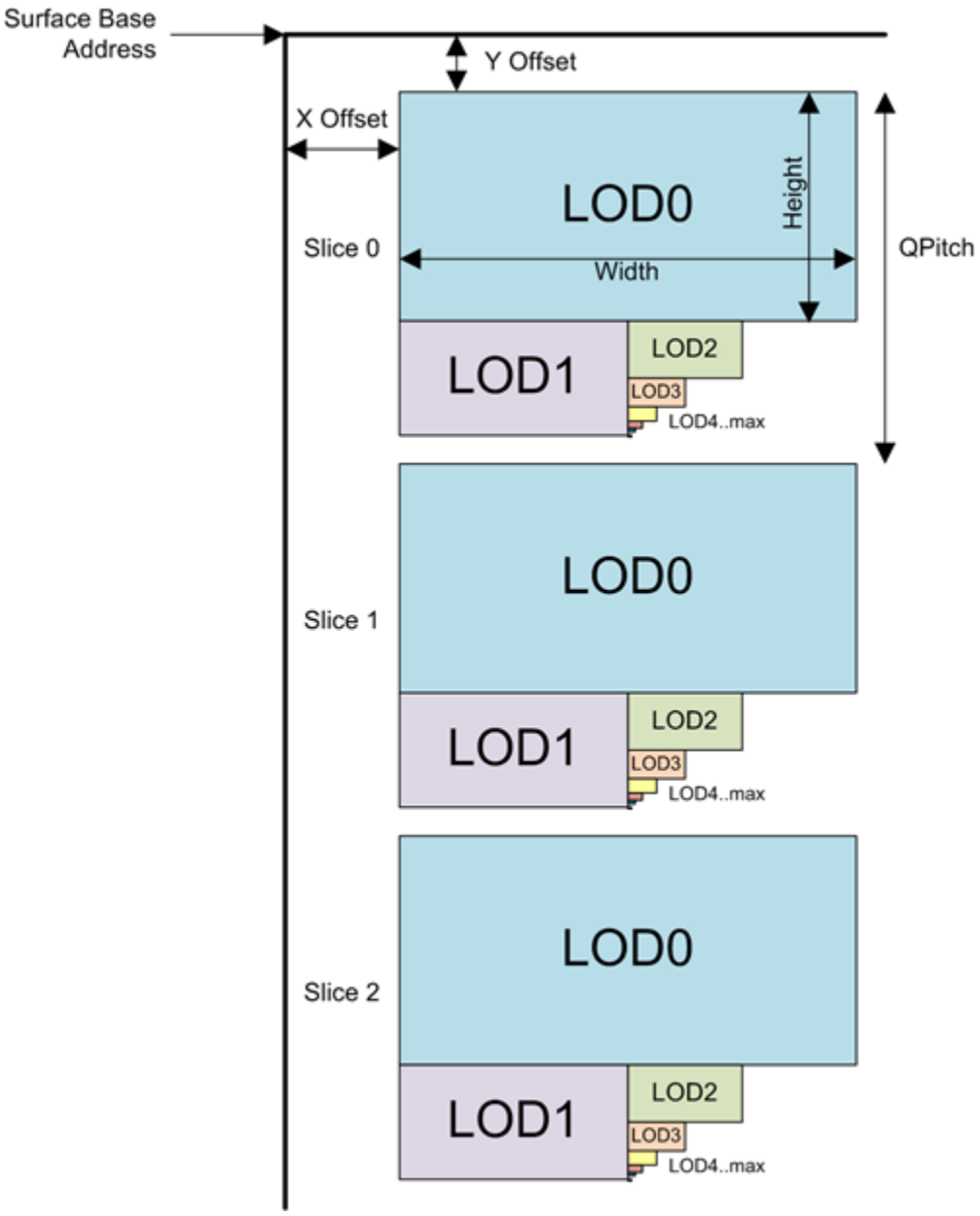
A surface can support multiple levels of details (LODs) or MIPs. The MIPCOUNT field in the RENDER_SURFACE_STATE defines how many MIPs a surface contains.

MIP0 or LOD0 is the largest, highest-detail MIP. The height, width and depth of this LOD is what is defined in the RENDER_SURFACE_STATE for that surface. Each subsequent

MIP is exactly one-half the height and width of the previous, making it 1/4th the size in memory.

The MIPs of a surface a layed out in memory using a 2-dimensional method as shown below. Volumetric and arrayed surfaces use multiple "slices" of this MIP layout, with each slice separted by QPITCH number of rows.

The diagram below shows many of the parameters of a 2D,2D Arrayed and 3D surface.



This 2-dimensional layout implies that there is padding required on the rows below LOD0 in order to ensure each row is the same number of texels.

If Tiling is enabled, then each MIP is layed out using one or more tiles. If TileYf or TileYs tiling is enabled (TR_MODE != NONE), then some of the MIPs may actually be stored in a MIPTail which fits in a single 64K or 4K tile. The layout above, then only applied to MIPs which are not packed in the MIP Tail.

Raw (Untyped) Buffers

Raw buffers also use the surface type of SURFTYPE_BUFFER, but the surface format is RAW. These buffers are one-dimensional. They are accessed with a single U parameter which is a byte offset into the buffer. Raw buffers are also supported only in linear memory.

The offset into the raw buffer is given directly by the U parameter.

```
Offset = U
```

Structured Buffers

A structured buffer is a surface type that is accessed by a 2-dimensional coordinate. It can be thought of as an array of structures, where each structure is a predefined number of DWords in size. The first coordinate (U) defines the array index, and the second coordinate (V) is a byte offset into the structure which must be a multiple of 4 (DWord-aligned). A structured buffer must be defined with **Surface Format** RAW.

The structured buffer has only one dimension programmed in SURFACE_STATE which indicates the array size. The byte offset dimension (V) is assumed to be bounded only by the **Surface Pitch**.

The two dimensional offset into the surface is defined directly by the U and V parameters. Structured buffers are linear.

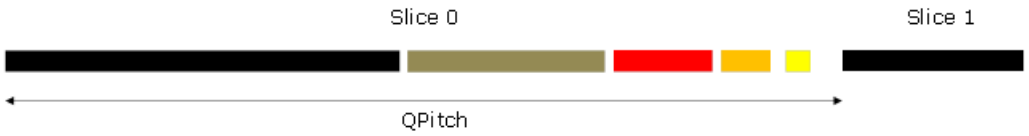
1D Surfaces

One-dimensional surfaces use a tiling mode of linear. Technically, they are not tiled resources, but the Tiled Resource Mode field in RENDER_SURFACE_STATE is still used to indicate the alignment requirements

for this linear surface (See 1D Alignement requirements for how 4K and 64KB Tiled Resource Modes impact alignment). Alternatively, a 1D surface can be defined as a 2D tiled surface (e.g. TileY or TileX) with

a height of 0.

Linear 1D surfaces are stored in a one-dimensional view of memory as follows:



Surface Pitch is ignored for 1D surfaces. **Surface QPitch** specifies the distance in pixels between array slices. QPitch should allow at least enough space for any mips that may be present.

A number of parameters are useful to determine where given pixels will be located on the 1D surface. First, the width for each LOD “L” is computed:

$$W_L = ((width >> L) > 0 ? width >> L : 1$$

Next, the aligned width parameter for each LOD "L" is computed. The "i" parameter is the horizontal alignment parameter set by a state field or defined as a constant, depending on the surface. The alignment parameter may change at one point in the mip chain based on **Mip Tail Start LOD**. The equation uses the I value that applies to the LOD being computed.

$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

Next, the offset to each LOD is determined. The offset has one dimension for 1D surfaces. The single element in the LOD_L vector is named LODUL.

$$LOD_0 = (0)$$
$$LOD_1 = (w_0)$$
$$LOD_2 = (w_0 + w_1)$$
$$LOD_3 = (w_0 + w_1 + w_2)$$
$$LOD_4 = (w_0 + w_1 + w_2 + w_3)$$
...

Based on the above parameters and the U and R (pixel address and array index, respectively), and the bytes per pixel of the surface format (Bpp), the offset "u" in bytes from the base address of the surface is given by:

```
u = [(R * QPitch) + LODUL + U] * Bpp
```

The layout documented in this section does not apply to sampler 1D surfaces if the tile mode is set to TileY or TileW.

Tiling and Mip Tail for 1D Surfaces

If tiling is enabled and **Tiled Resource Mode** is set to TRMODE_NONE, the above sequence applies throughout the full range of LODs. For other settings of Tiled Resource Mode, the mip tail offset is given by the following, where S is the **Mip Tail Start LOD**:

$$LOD_S = (w_0+w_1+w_2+w_3+ \ldots + w_{S-1})$$

The mip tail exhibits a different arrangement than the rest of the surface. LODs are aligned to the left side of the space available.

The offsets *into the mip tail block* are given by the following table for each LOD in the mip tail. The offsets given here need to be added to the LOD_s offset computed earlier to obtain the offset into the surface LOD_L.

TRMODE_64KB LOD	TRMODE_4KB LOD	128 bpe	64 bpe	32 bpe	16 bpe	8 bpe
s		(2048)	(4096)	(8192)	(16384)	(32768)
s+1		(1024)	(2048)	(4096)	(8192)	(16384)
s+2		(512)	(1024)	(2048)	(4096)	(8192)
s+3		(256)	(512)	(1024)	(2048)	(4096)
s+4	s	(128)	(256)	(512)	(1024)	(2048)

TRMODE_64KB LOD	TRMODE_4KB LOD	128 bpe	64 bpe	32 bpe	16 bpe	8 bpe
s+5	s+1	(64)	(128)	(256)	(512)	(1024)
s+6	s+2	(48)	(96)	(192)	(384)	(768)
s+7	s+3	(32)	(64)	(128)	(256)	(512)
s+8	s+4	(28)	(56)	(112)	(224)	(448)
s+9	s+5	(24)	(48)	(96)	(192)	(384)
s+10	s+6	(20)	(40)	(80)	(160)	(320)
s+11	s+7	(16)	(32)	(64)	(128)	(256)
s+12	s+8	(12)	(24)	(48)	(96)	(192)
s+13	s+9	(8)	(16)	(32)	(64)	(128)
s+14	s+10	(4)	(8)	(16)	(32)	(64)
s+15	s+11	(0)	(0)	(0)	(0)	(0)

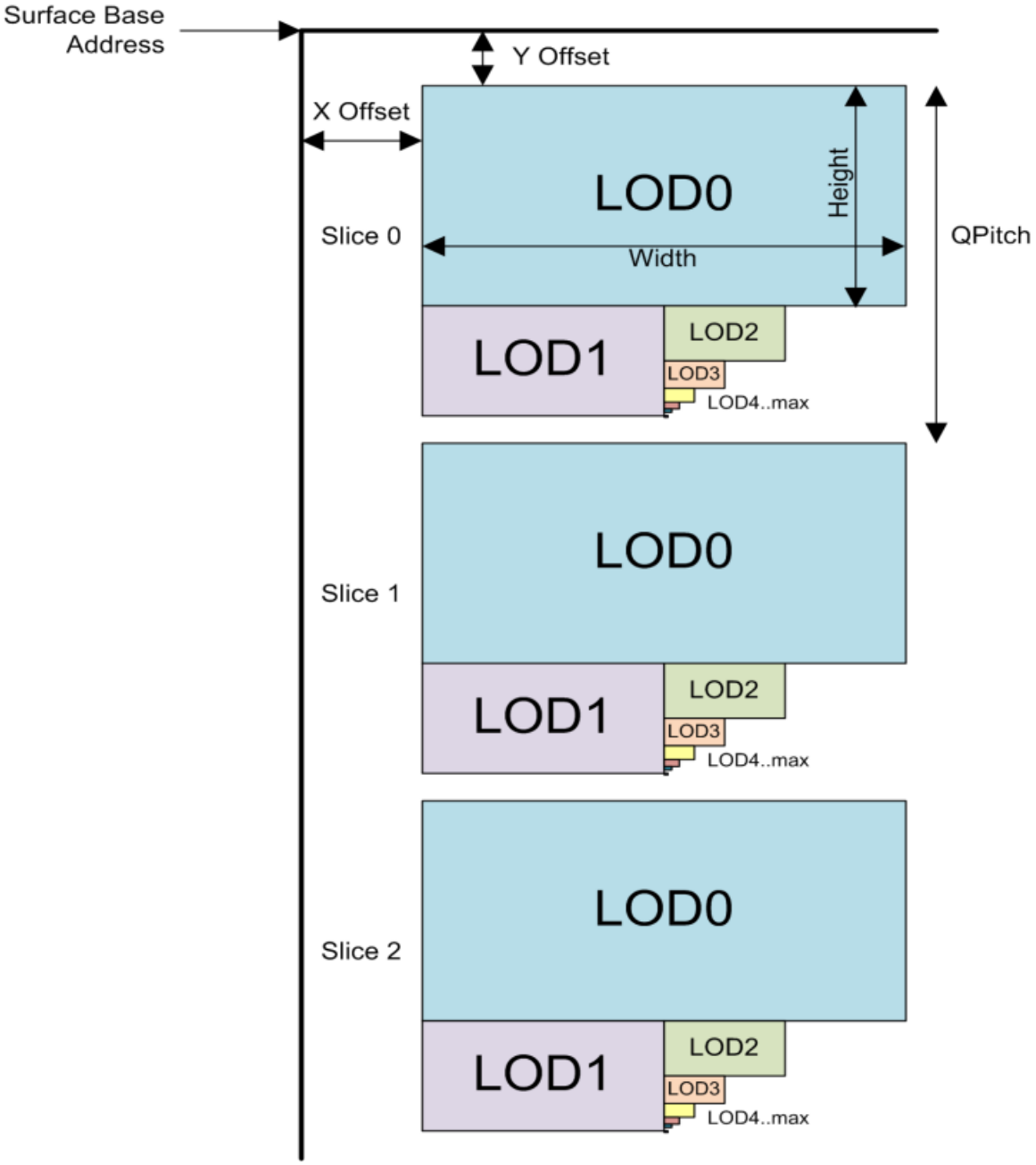
1D Alignment Requirements

The horizontal alignment field in the RENDER_SURFACE_STATE are ignored for standard tiling formats (TRMODE = NONE). In the case of standard tiling formats the alignment requirements are fixed and are provided for by the tables below for 1D surfaces.

Tiled Resource Mode	Bits per Element	Horizontal Alignment
TRMODE_64KB	128	4096
	64	8192
	32	16384
	16	32768
	8	65536
TRMODE_4KB	128	256
	64	512
	32	1024
	16	2048
	8	4096
TRMODE_NONE	Any	64

2D Surfaces

2D surfaces represent two-dimensional bitmaps, which can also be mip-mapped and/or consist of array slices, effectively representing multiple 2D sub-surfaces within a single surface. The diagram below shows many of the parameters of a 2D surface or Arrayed 2D Surface and what they mean.



All surface parameters are defined in terms of texels (agnostic to the surface format).

Surface Pitch defines the distance in bytes between rows of the surface and is a function of the **Width** of LOD. **QPitch** specifies the distance in rows between array slices and is a function of the **Height**. **QPitch** should allow at least enough space for any Mips that may be present.

There can also be non-zero offsets (**X_Offset** and **Y_Offset)** defined from the base address which can be used to provide padding or provide an offset to a lower-detail LOD.

There are limitations to the physical size of an LOD in the sampler texture cache. An LOD must be aligned to a cache-line except for some special cases related to Planar YUV surfaces. In general, the cache-alignment restriction implies there is a minimum height for an LOD of 4 texels. So, LODs which are smaller than 4 high are padded. For tiled surfaces with **TR_MODE** != TR_NONE this restriction is not significant because the MIP tail will be used for smaller MIPs and the slots are a minimum of 64B. For non-tiled surface or surfaces where **TR_MODE == TR_NONE**, Mips smaller than 4 high start at the top of the region, and they are padded. This padding leads to a case where the smallest LOD starts "below" LOD1 vertically.

Calculating Texel Location

This section describes how the texel location is calculated once the Surface State and LOD are known. A number of parameters are useful to determine where given pixels are located on the 2D surface. The width ($\mathbf{W}_L$) and height ($\mathbf{H}_L$) for each LOD "L" is computed by the formula:

W_L = ((width»L) > 0 ? width»L:1)

H_L = ((height»L) > 0? height»L:1)

The LOD width and height for each subsequent LOD is one-half the previous LOD, with the minimum dimension being 1 texel. If the surface is multisampled and it is a depth or stencil surface or **Multisampled Surface Storage Format** in SURFACE_STATE is MSFMT_DEPTH_STENCIL, $\mathbf{W}_L$ and $\mathbf{H}_L$ must be adjusted as follows:

Number of Multisamples	WL =	HL =
2	ceiling(W_L / 2) * 4	H_L [no adjustment]
4	ceiling(W_L / 2) * 4	ceiling(H_L / 2) * 4
8	ceiling(W_L / 2) * 8	ceiling(H_L / 2) * 4
16	ceiling(W_L / 2) * 8	ceiling(H_L / 2) * 8

Next, aligned width, height, and depth parameters for each LOD "L" must be computed. The "i" and "j" parameters are horizontal and vertical alignment parameters set by state fields or defined as constants, depending on the surface. Depth has no alignment parameter (effectively it is 1).

The equation uses the i and j values that apply to the LOD being computed. The "p" and "q" parameters define the width and height in texels of the compression block for compressed surface formats. Both p and q are defined to equal 1 for uncompressed surface formats.

$$w_L = i * p * ceil\left(\frac{W_L}{i * p}\right)$$

$$h_L = j * q * ceil\left(\frac{H_L}{j * q}\right)$$

Once the height (h_i) and width (w_i) of each LOD is computed , the offset to each LOD can be determined. The offset is a vector with two dimensions. The elements in the LOD_L vector are named in order $LODU_L$, $LODV_L$.

LOD offset computation for **TR_MODE** == TR_NONE or when L < **Mip Tail Start LOD**:

LOD_0 = (0,0)

LOD_1 = (0,h_0)

LOD_2 = (w_1,h_0)

LOD_3 = (w_1,h_0 + h_2)

LOD_4 = (w_1,h_0 + h_2 + h_3)

…

LOD_N = (w_1, h_0 + h_2 + h_3 … + h_{N-1})

Where N = MIP_COUNT for the surface. As noted previous in this section, the value of h_2 + h_3… + h_{N-1} may be greater than h_1 due to alignment requirements.

Based on the above parameters and the U, V, and R (two dimensional pixel address U/V and array index R), and the *bytes* per pixel of the surface format (Bpp), the offsets u in bytes and v in rows are given by:

u = (U + $LODU_L$) * Bpp

v = (R * QPitch) + $LODV_L$ + V

For a description of how the Mip Tail is laid out and offsets into the Mip Tail are calculated see the sub-section on 2D Surface Layout for Mip Tails.

The two dimensional offset into the surface (for non-MipTail cases) is defined by the u and v values computed above. The lower virtual address bits are determined by the following table, based on the bits of u and v. An *element* is defined as a pixel for uncompressed surface formats and a compression block for compressed surface formats. Empty bit positions indicate that the bit is not part of the tile swizzle and is filled in with equations given next (note that linear mode has all bits empty—there is no swizzling in linear mode).

Tile Mode	Bits per Element	TileID constants		Virtual Address Bits															
		Cv	Cu	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TileYS	64 & 128	6	10	u9	v5	u8	v4	u7	v3	u6	v2	u5	u4	v1	v0	u3	u2	u1	u0
	16 & 32	7	9	u8	v6	u7	v5	u6	v4	u5	v3	u4	v2	v1	v0	u3	u2	u1	u0
	8	8	8	u7	v7	u6	v6	u5	v5	u4	v4	v3	v2	v1	v0	u3	u2	u1	u0
TileYF	64 & 128	4	8					u7	v3	u6	v2	u5	u4	v1	v0	u3	u2	u1	u0
	16 & 32	5	7					u6	v4	u5	v3	u4	v2	v1	v0	u3	u2	u1	u0
	8	6	6					u5	v5	u4	v4	v3	v2	v1	v0	u3	u2	u1	u0
TileY	all	5	7					u6	u5	u4	v4	v3	v2	v1	v0	u3	u2	u1	u0
TileX	all	3	9					v2	v1	v0	u8	u7	u6	u5	u4	u3	u2	u1	u0
TileW	all	6	6					u5	u4	u3	v5	v4	v3	v2	u2	v1	u1	v0	u0
Linear	all	0	0																

The TileID fills the upper bits of the virtual address (starting with the lowest blank bit in the above table):

TileID = (v » Cv) * (Pitch » Cu) + (u » Cu)

Where Pitch is the **Surface_Pitch** field from RENDER_SURFACE_STATE.

Note: Multisampled CMS and UMS surfaces use a modified address bit swizzling table rather than the one above. Refer to the *Multisampled*2D Surfaces section for details.

Tiling and Mip Tails for 2D Surfaces

When surface is Tiled (Tile_Mode=YMAJOR) and Tile Resources are enabled (TR_MODE != TR_NONE), a 2D surface can contain a Mip Tail for smaller Mip sizes.

When LOD (L) is less than the **Mip Tail Start LOD** (S) declared in the Surface State the offset to the start of LOD is calculated as shown above.

If the LOD is greater than or equal to **Mip Tail Start LOD** field in the surface state then the MIP Tail layout below is used..

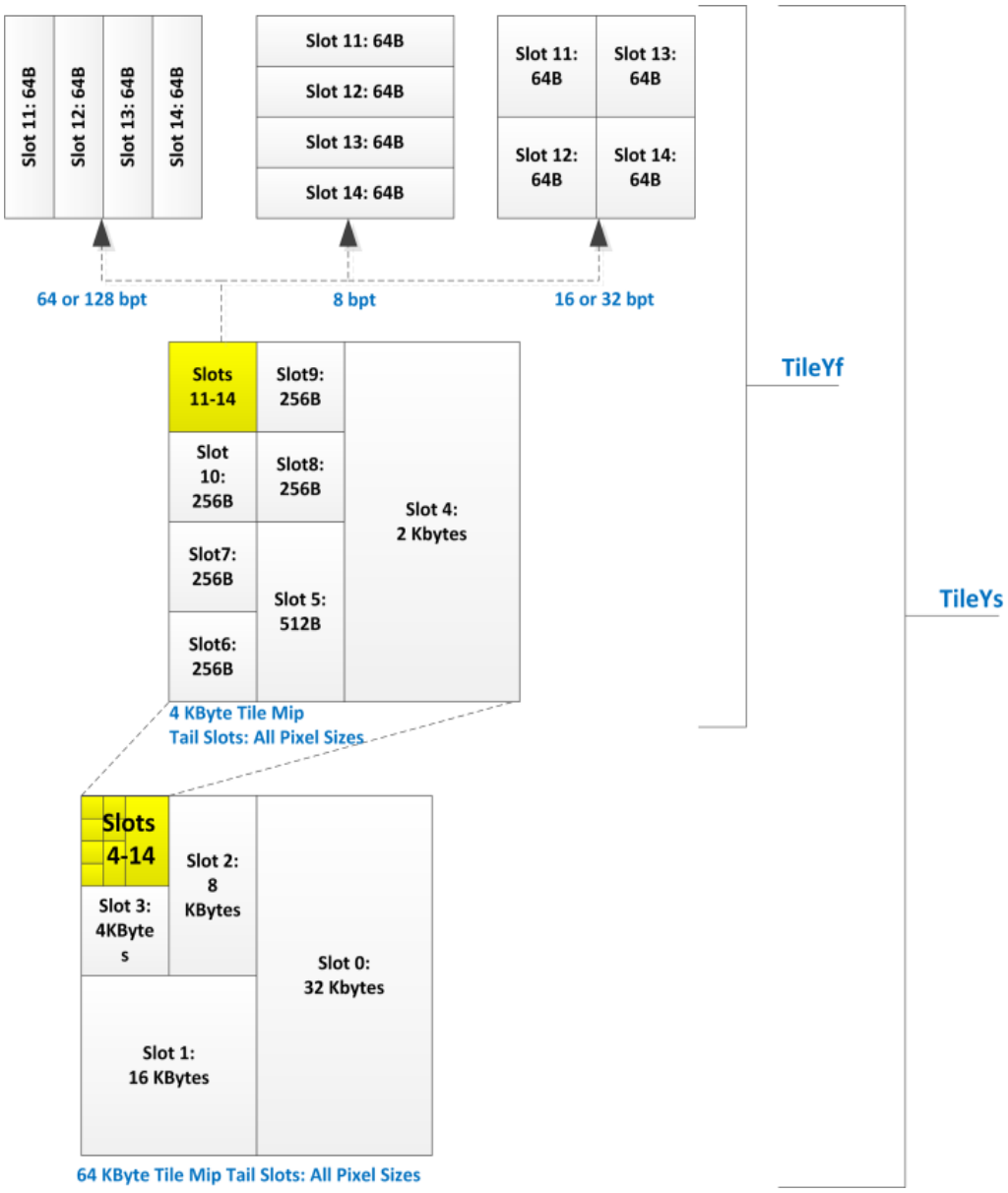
For tiled resources, the mip tail offset is given by the following, where s is the Mip Tail Start LOD:

$LOD_S = (w_1, h_0 + h_2 + h_3 + ... + h_{S-1})$

The LOD's in the Mip Tail are arranged differently than the other LOD's.

The diagram below shows the 64KB TileYS Mip Tail layout of LODs within it, with "slots" indicating the LOD contained within (slot 0 corresponds to LOD_s above). LOD's are aligned to the upper left corner of the space available. The block marked "Slots 4-14" is a 4KB tile arrangement as shown. Within this 4KB tile slots 11 thru 14 are arranged differently depending on the number bits per texel (bpt).

A TileYf (4KByte) Mip Tail will start with the 4KByte tile shown, but the slots will be renumbered to start at Slot0 rather than Slot4. The layout of slots 11 through 14 remain the same. Note that Slots 12-14 are NOT 256-Byte aligned which is not compliant with the standard MIP Tail layout. These slots are not supported for Standard Tiling.



The offsets *into the Mip Tail tile* are given by the following table for each LOD in the Mip tail. Each entry in the table is a horizontal (M_U) and vertical (M_V) position (in texels) from the upper left corner of the Mip Tail. If LOD >= S (starting LOD for MIP Tail), then these Mip Tail offsets must be added to the $LODU_L$ and $LODV_L$ calculated above.

Note that many of the higher LODs are not possible given surface size constraints, but they are listed here for reference. The offsets given here need to be added to the LODs offset computed earlier to obtain the offset into the surface LOD_L.

TileYS LOD					TileYF LOD	128 bpe	64 bpe	32 bpe	16 bpe	8 bpe
1x	2x	4x	8x	16x		(M_U,M_V)	(M_U,M_V)	(M_U,M_V)	(M_U,M_V)	(M_U,M_V)
s						(32,0)	(64,0)	(64,0)	(128,0)	(128,0)
s+1	s					(0,32)	(0,32)	(0,64)	(0,64)	(0,128)
s+2	s+1	s				(16,0)	(32,0)	(32,0)	(64,0)	(64,0)
s+3	s+2	s+1	s			(0,16)	(0,16)	(0,32)	(0,32)	(0,64)
s+4	s+3	s+2	s+1	s	s	(8,0)	(16,0)	(16,0)	(32,0)	(32,0)
s+5	s+4	s+3	s+2	s+1	s+1	(4, 8)	(8, 8)	(8, 16)	(16, 16)	(16, 32)
s+6	s+5	s+4	s+3	s+2	s+2	(0, 12)	(0, 12)	(0, 24)	(0, 24)	(0, 48)
s+7	s+6	s+5	s+4	s+3	s+3	(0, 8)	(0, 8)	(0, 16)	(0, 16)	(0, 32)
s+8	s+7	s+6	s+5	s+4	s+4	(4, 4)	(8, 4)	(8, 8)	(16, 8)	(16, 16)
s+9	s+8	s+7	s+6	s+5	s+5	(4, 0)	(8, 0)	(8, 0)	(16, 0)	(16, 0)
s+10	s+9	s+8	s+7	s+6	s+6	(0, 4)	(0, 4)	(0, 8)	(0, 8)	(0, 16)
s+11	s+10	s+9	s+8	s+7	s+7	(3, 0)	(6, 0)	(4, 4)	(8, 4)	(0, 12)
s+12	s+11	s+10	s+9	s+8	s+8	(2, 0)	(4, 0)	(4, 0)	(8, 0)	(0, 8)
s+13	s+12	s+11	s+10	s+9	s+9	(1, 0)	(2, 0)	(0, 4)	(0, 4)	(0, 4)
s+14	s+13	s+12	s+11	s+10	s+10	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)

If the LOD is located in the MIP Tail then the equation for calculating the byte positions for u and v become:

```
u = (U + LODUS + MU) * Bpp
v = (R * QPitch) + LODVS + MV + V
```

where M_U and M_V are the offset parameters from the table above for the given slot in the MIP Tail.

Programming Note	
Context:	Lossless Compression and MIP Tail
Lossless compression must not be used on surfaces which have MIP Tail which contains MIPs for Slots greater than 11.	

2D/CUBE Alignment Requirement

The vertical and horizontal alignment fields in the RENDER_SURFACE_STATE are ignored for standard tiling formats (TRMODE = NONE). In the case of standard tiling formats the alignment requirements are fixed and are provided for by the tables below for 2D and CUBE surface.

Tile Mode	Bits per Element	Horizontal Alignment	Vertical Alignment
TileYS	128	64	64
	64	128	64
	32	128	128
	16	256	128
	8	256	256
TileYF	128	16	16
	64	32	16
	32	32	32
	16	64	32
	8	64	64

For MSFMT_MSS type multi-sampled TileYS surfaces, the alignments given above must be divided by the appropriate value from the table below.

Number of Multisamples	Horizontal Alignment is divided by	Vertical Alignment is divided by
2	2	1
4	2	2
8	4	2
16	4	4

Multisampled 2D Surfaces

There are three types of multisampled surface layouts designated as follows:

- **IMS** Interleaved Multisampled Surface

• **CMS** Compressed Mulitsampled Surface

• **UMS** Uncompressed Multisampled Surface

These surface layouts are described in the following sections.

Interleaved Multisampled Surfaces

IMS surfaces are supported in all generations for depth and stencil surfaces. These surfaces contain the samples in an interleaved fashion, with the underlying surface in memory having a height and width that is larger than the non-multisampled surface as follows:

- 4x MSAA: 2x width and 2x height of non-multisampled surface.
- 8x MSAA: 4x width and 2x height of non-multisampled surface.
- 16x MSAA: 4X width and 4X height of the non-multisampled surface.

When sampling from an IMS surface (e.g. ld2dms), the coordinates are automatically scaled to handle the increased physical size of the map.

Compressed Multisampled Surfaces

Multisampled render targets can be compressed. If **Auxiliary Surface Mode** in SURFACE_STATE is set to AUX_CCS, hardware handles the compression using a software-invisible algorithm. However, performance optimizations in the multisample resolve kernel using the sampling engine are possible if the internal format of these surfaces is understood by software. This section documents the formats of the Multisample Control Surface (MCS) and Multisample Surface (MSS).

MCS Surface

The MCS surface consists of one element per pixel, with the element size being an 8-bit unsigned integer value for 4x multisampled surfaces, a 32-bit unsigned integer value for 8x multisampled surfaces and a 64-bit unsigned integer value for 16x multisampled surfae. Each field within the element indicates which sample slice (SS) the sample resides on.

2x MCS

The 2x MCS is 8 bits per pixel. The 8 bits are encoded as follows:

7:2	1	0
reserved	sample 1 SS	sample 0 SS

Each 1-bit field indicates which sample slice (SS) the sample's color value is stored. An MCS value of 0x00 indicates that both samples are stored in sample slice 0 (thus have the same color). This is the fully compressed case. An MCS value of 0x03 indicates that all samples in the pixel are in the clear state, and none of the sample slices are valid. The pixel's color must be replaced with the surface's clear value.

4x MCS

The 4x MCS is 8 bits per pixel. The 8 bits are encoded as follows:

7:6	5:4	3:2	1:0
sample 3 SS	sample 2 SS	sample 1 SS	sample 0 SS

Each 2-bit field indicates which sample slice (SS) the sample's color value is stored. An MCS value of 0x00 indicates that all four samples are stored in sample slice 0 (thus all have the same color). This is the fully compressed case. An MCS value of 0xff indicates that all samples in the pixel are in the clear state, and none of the sample slices are valid. The pixel's color must be replaced with the surface's clear value. See the section below on Clear Pixel Conditions for additional encoding information.

8x MCS

Extending the mechanism used for the 4x MCS to 8x requires 3 bits per sample times 8 samples, or 24 bits per pixel. The 24-bit MCS value per pixel is placed in a 32-bit footprint, with the upper 8 bits unused as shown below. See the section below on Clear Pixel Conditions for additional encoding information.

31:24	23:21	20:18	17:15	14:12	11:9	8:6	5:3	2:0
reserved (MBZ)	sample 7 SS	sample 6 SS	sample 5 SS	sample 4 SS	sample 3 SS	sample 2 SS	sample 1 SS	sample 0 SS

16x MCS

The 16x MCS is 64 bits per pixel. The 64 bits are encoded as follows:

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32
sample 15 SS	sample 14 SS	sample 13 SS	sample 12 SS	sample 11 SS	sample 10 SS	sample 9 SS	sample 8 SS

31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
sample 7 SS	sample 6 SS	sample 5 SS	sample 4 SS	sample 3 SS	sample 2 SS	sample 1 SS	sample 0 SS

Other than this, the 16x algorithm is the same as the 8x algorithm. The MCS value indicating clear state is 0xffffffff_ffffffff. See the section below on Clear Pixel Conditions for additional encoding information.

Clear Pixel Conditions

The MCS format allows for the encoding of clear value for one or more planes of the multi-sampled surface. A value of all 1's for defined MCS bits indicates that all planes of the multi-sampled surface are clear. For example, a value of 0x3 for 2X MSAA MCS byte means that both planes of the pixel are clear. Likewise a value of 0xff for X4, 0xffffffff for X8 and 0xffffffff_ffffffff for X16 MSAA means that all planes of the pixel are clear.

In the case where not all planes are clear, but at least 2 planes are clear the encoding of the MCS given above is changed. If the MCS value for plane 0 is non-zero, then all planes which are at all 1's are clear and all other planes are referencing the plane indicated by their respective MCS value minus 1. For example, a 4X MSAA MCS value of 01 10 01 11 means that MCS 0 and 2 are referencing plane 0, and MCS 1 is referencing plane 1, and MCS 3 is clear.

MSS Surface

The physical MSS surface is stored identically to a 2D array surface, with the height and width matching the pixel dimensions of the logical multisampled surface. The number of array slices in the physical surface is 2, 4, 8, or 16 times that of the logical surface (depending on the number of multisamples). Sample slices belonging to the same logical surface array slice are stored in adjacent physical slices. The sampling engine ld2dss message gives direct access to a specific sample slice.

Tiling for CMS and UMS Surfaces

Multisampled CMS and UMS use a modified table from non-mulitsampled 2D surfaces.

TileY, TileX, TileW, and Linear: Treat as 2D array, with the array index "R" modified as follows. "n" is the number of multisamples, "ss" is the sample slice index with range 0..n-1.

```
R(new) = ( R(old) « log2(n) ) | ss
```

TileYS: In addition to u and v, the sample slice index "ss" is included in the address swizzling according to the following table. Because of this, the mip tail holds one less LOD for each successive number of multisamples. Refer to the mip tail table in the previous section for behavior of the mip tail for each number of multisamples.

Number of Multisamples	Bits per Element	TileID constants		Virtual Address Bits									
		Cv	Cu	15	14	13	12	11	10	9	8	7	6
2x	64 & 128	6	9	ss0	v5	u8	v4	u7	v3	u6	v2	u5	u4
	16 & 32	7	8	ss0	v6	u7	v5	u6	v4	u5	v3	u4	v2
	8	8	7	ss0	v7	u6	v6	u5	v5	u4	v4	v3	v2
4x	64 & 128	5	9	ss1	ss0	u8	v4	u7	v3	u6	v2	u5	u4
	16 & 32	6	8	ss1	ss0	u7	v5	u6	v4	u5	v3	u4	v2
	8	7	7	ss1	ss0	u6	v6	u5	v5	u4	v4	v3	v2
8x	64 & 128	5	8	ss2	ss1	ss0	v4	u7	v3	u6	v2	u5	u4
	16 & 32	6	7	ss2	ss1	ss0	v5	u6	v4	u5	v3	u4	v2
	8	7	6	ss2	ss1	ss0	v6	u5	v5	u4	v4	v3	v2
16x	64 & 128	4	8	ss3	ss2	ss1	ss0	u7	v3	u6	v2	u5	u4
	16 & 32	5	7	ss3	ss2	ss1	ss0	u6	v4	u5	v3	u4	v2
	8	6	6	ss3	ss2	ss1	ss0	u5	v5	u4	v4	v3	v2

Note that Cv and Cu are also different that the values for non-multisampled 2D surfaces.

Uncompressed Multisampled Surfaces

UMS surfaces similar to CMS, except that the **Auxiliary Surface Mode** is set to AUX_NONE, meaning that there is no MCS surface. UMS contains only an MSS surface, where each sample is stored on its sample slice (SS) of the same index.

Quilted Textures

A quilted texture is a 2D texture made up of quilt slices, each of which is a portion of the surface up to 16k x 16k texels in size. The quilt slices themselves are organized in a matrix up to 32 x 32. "Quilt Width" and "Quilt Height" fields indicate the dimensions of the surface in quilt slices. "Height" and "Width" fields indicate the size of each quilt slice in texels. The total size of the quilted texture can be up to 512k x 512k texels.

In addition, arrays of quilted textures are supported. The total number of array slices is limited to 2048 / (QuiltWidth * QuiltHeight).

A surface is defined as a "Quilted Texture" if either the "Quilt Width" or "Quilt Height" field in SURFACE_STATE is nonzero. A quilted texture is stored in the storage format as a 2D array, with each quilt square occupying one array slice. The following equation indicates how the array slice is computed from the Qx, Qy, and R parameters, where Qx and Qy are the quilt slice coordinates and R is the array index.

```
ArraySliceIndex = (R * QuiltWidth * QuiltHeight) + Qy * QuiltWidth + Qx
```

Quilted textures do NOT support wrapping. U and V coordinates must be in the range of [0.0,1.0).

Quilted textures can only be supported on tiled surfaces (e.g. TileY, TileYs, TileYf).

Cube Surfaces

The 3D Pipeline supports *cubic environment maps*, conceptually arranged as a cube surrounding the origin of a 3D coordinate system aligned to the cube faces. These maps can be used to supply texel (color/alpha) data of the environment in any direction from the enclosed origin, where the direction is supplied as a 3D "vector" texture coordinate. These cube maps can also be mipmapped.

Each texture map level is represented as a group of six, square *cube face* texture surfaces. The faces are identified by their relationship to the 3D texture coordinate system. The subsections below describe the cube maps as described at the API as well as the memory layout dictated by the hardware.

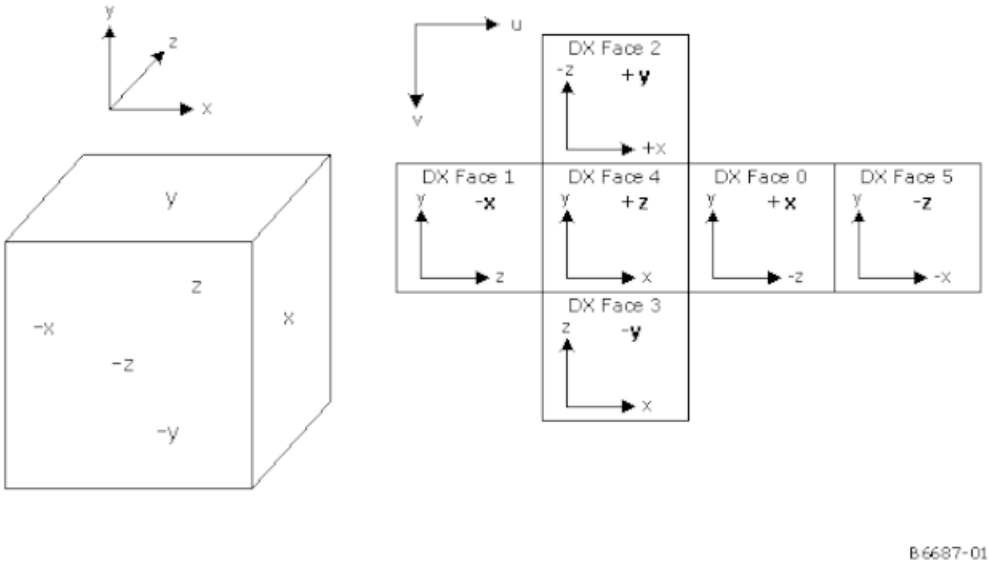
The diagram below describes the cube map faces as they are defined at the DirectX API. It shows the axes on the faces as they would be seen from the inside (at the origin).

Programming Note	
Context:	Cube Maps and Tiling
Cube Maps with linear (non-tiled) layout are allowed ONLY with Texture Coordinate Mode of TEXCOORDMODE_CUBE or TEXCOORDMODE_CLAMP.	

The origin of the U,V texel grid is at the top left corner of each face.

This will be looking directly at face 4, the +z –face. Y is up by default.

DirectX Cube Map Definition



Each face has a corresponding face identifier “f” as indicated in the following table:

face	face identifier “f”
+x	0
-x	1
+y	2
-y	3
+z	4
-z	5

A cube surface is stored in memory the same as a 2D array, with the face identifier “f” and array index “ai” being transformed into the “R” coordinate used in storing 2D arrays using the following equation:

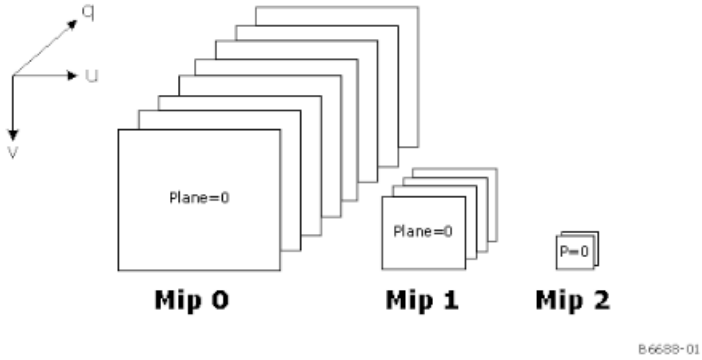
```
R = (ai * 6) + f
```

Refer to the “2D Surfaces” section for details on how 2D arrays are stored.

3D Surfaces

Multiple texture map surfaces (and their respective mipmap chains) can be arranged into a structure known as a Texture3D (volume) texture. A volume texture map consists of many *planes* of 2D texture maps. See *Sampler* for a description of how volume textures are used.

Volume Texture Map



Surface Pitch defines the distance in bytes between rows of the surface. **Surface QPitch** specifies the distance in rows between R-slices. QPitch should allow at least enough space for any mips that may be present.

A number of parameters are useful to determine where given pixels are located on the 3D surface. First, the width, height, and depth for each LOD "L" is computed:

$$W_L = ((width >> L) > 0\,?\,width >> L : 1)$$
$$H_L = ((height >> L) > 0\,?\,height >> L : 1)$$
$$D_L = ((depth >> L) > 0\,?\,depth >> L : 1)$$

Next, aligned width, height, and depth parameters for each LOD "L" are computed. The "i", "j", and "k" parameters are the horizontal, vertical, and depth alignment parameters set by state fields or defined as constants. The alignment parameters may change at one point in the mip chain based on **Mip Tail Start LOD**. The equation uses the i/j values that apply to the LOD being computed. The "p", "q", and "s" parameters define the width, height, and depth in texels of the compression block for compressed surface formats. These are all defined to equal 1 for uncompressed surface formats.

$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$
$$h_L = j * ceil\left(\frac{H_L}{j}\right)$$
$$d_L = D_L$$
$$d_L = k * s * ceil\left(\frac{D_L}{k * s}\right)$$

Next, the offset to each LOD is determined. The offset is a vector with three dimensions. The elements in the LOD_L vector are named in order $LODU_L$, $LODV_L$, $LODR_L$.

LOD offset computation for **Tiled Resource Mode** == TR_NONE or when L < **Mip Tail Start LOD**:

$LOD_0 = (0,0,0)$
$LOD_1 = (0,h_0,0)$
$LOD_2 = (w_1,h_0,0)$
$LOD_3 = (w_1,h_0+h_2,0)$
$LOD_4 = (w_1,h_0+h_2+h_3,0)$
...

For the Primary Surface

Based on the above parameters and the U, V, and R (three dimensional pixel address), and the bytes per pixel of the surface format (Bpp), the offsets u in bytes, v in rows, and r in slices are given by:

```
u = [U + LODUL] * Bpp
v = LODVL + V
r = LODRL + R
```

The three dimensional offset into the surface is defined by the u, v, and r values computed above. The lower virtual address bits are determined by the following table, based on the bits of u, v, and r. An *element* is defined as a pixel for uncompressed surface formats and a compression block for compressed surface formats.

Empty bit positions indicate that the bit is not part of the tile swizzle and is filled in with the equations given next (note that linear mode has all bits empty—there is no swizzling in linear mode).

The TileID fills the upper bits of the virtual address (starting with the lowest blank bit in the above table):

```
TileID = [(r » Cr) * (QPitch » Cv) + (v » Cv)] * (Pitch » Cu) + (u » Cu)
```

For the CCS Auxiliary Surface

The CCS is stored differently for the 3D surface type. CCS supports only TileY tile mode, which does not have a three dimensional offset. Instead, the 3D CCS follows a scheme similar to 2D surfaces. Based on the above parameters and the U, V, and R (three dimensional pixel address, shifted to adjust for control block size in bytes), the offsets u in bytes and v in rows are given by:

```
u = [U + LODUL]
v = (R * QPitch) + LODVL + V
```

The two dimensional offset into the surface is defined by the u and v values computed above. The lower virtual address bits are determined by the following table, based on the bits of u and v.

Empty bit positions indicate that the bit is not part of the tile swizzle and is filled in with equations given next.

Tile Mode	Bits per Element	TileID constants			Virtual Address Bits														
		Cr	Cv	Cu	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TileY	5	7							u6	u5	u4	v4	v3	v2	v1	v0	u3	u2	u1	u0

The TileID fills the upper bits of the virtual address (starting with the lowest blank bit in the above table):

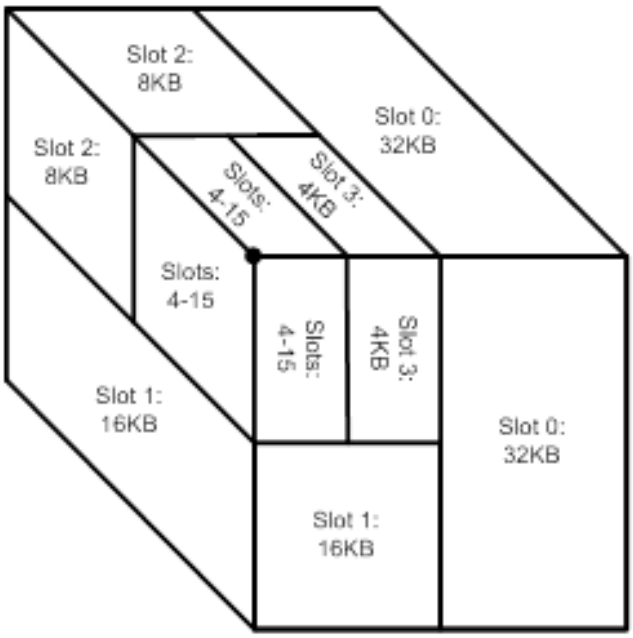
```
TileID = (v » Cv) * (Pitch » Cu) + (u » Cu)
```

Tiling and Mip Tails for 3D Surfaces

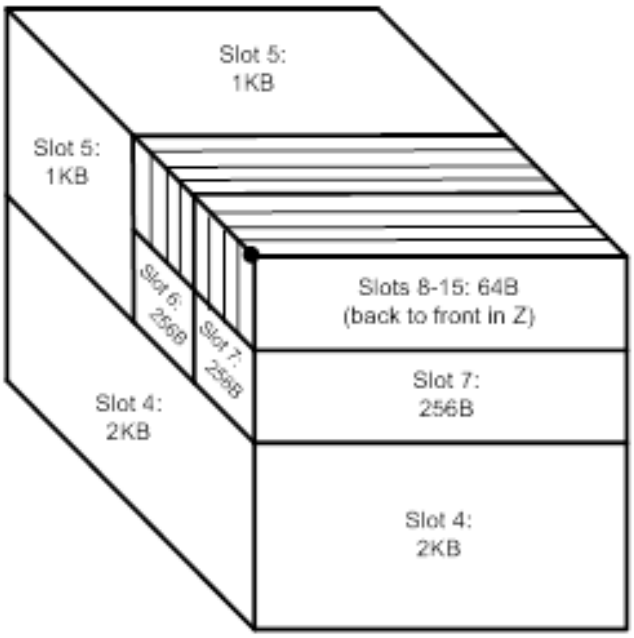
For tiled surfaces where Tiled Resource Mode != TR_NONE, the surface may contain a mip tail. The Mip tail offset is given by the following, where S is the Mip Tail Start LOD:

$$LOD_S = (w_1,h_0+h_2+h_3+ \dots + h_{S-1},0)$$

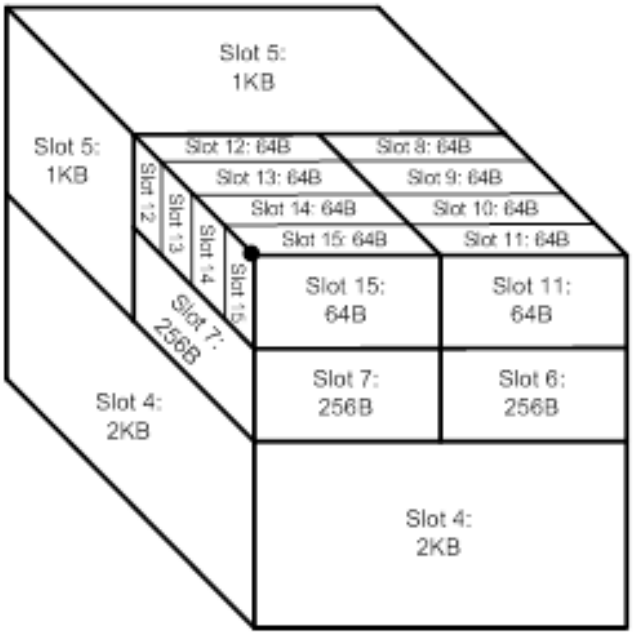
The mip tail exhibits a different arrangement than the rest of the surface. The diagram below shows the 64KB TileYS mip tail and the arrangement of LODs within it, with “slots” indicating the LOD contained within (slot 0 corresponds to LOD s). LODs are aligned to the front upper left corner of the space available. The block marked “Slots 4-15” contains one of the 4KB tile arrangements within, depending on the surface format bits per element. For TileYF, only the 4KB tile exists, with 4 subtracted from each slot number.



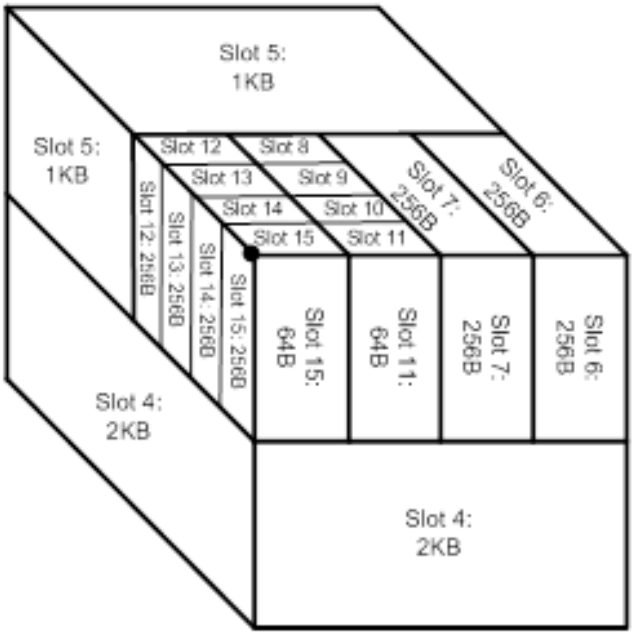
64KB Tile Mip Tail Slots:
All Pixel Depths



4KB Tile Mip Tail Slots:
8-bit & 16-bit Pixels



4KB Tile Mip Tail Slots:
32-bit Pixels



4KB Tile Mip Tail Slots:
64-bit & 128-bit Pixels

The offsets into the mip tail tile are given by the following table for each LOD in the mip tail. Note that many of the higher LODs are not possible given surface size constraints, but they are listed here for reference. The offsets given here need to be added to the LODs offset computed earlier to obtain the offset into the surface LOD_L.

TileYS LOD	TileYF LOD	128 bpe	64 bpe	32 bpe	16 bpe	8 bpe
s		(8, 0, 0)	(16, 0, 0)	(16, 0, 0)	(16, 0, 0)	(32, 0, 0)
s+1		(0, 8, 0)	(0, 8, 0)	(0, 16, 0)	(0, 16, 0)	(0, 16, 0)
s+2		(0, 0, 8)	(0, 0, 8)	(0, 0, 8)	(0, 0, 16)	(0, 0, 16)
s+3		(4, 0, 0)	(8, 0, 0)	(8, 0, 0)	(8, 0, 0)	(16, 0, 0)
s+4	s	(0, 4, 0)	(0, 4, 0)	(0, 8, 0)	(0, 8, 0)	(0, 8, 0)
s+5	s+1	(0, 0, 4)	(0, 0, 4)	(0, 0, 4)	(0, 0, 8)	(0, 0, 8)
s+6	s+2	(3, 0, 0)	(6, 0, 0)	(4, 4, 0)	(0, 4, 4)	(0, 4, 4)
s+7	s+3	(2, 0, 0)	(4, 0, 0)	(0, 4, 0)	(0, 4, 0)	(0, 4, 0)
s+8	s+4	(1, 0, 3)	(2, 0, 3)	(4, 0, 3)	(0, 0, 7)	(0, 0, 7)
s+9	s+5	(1, 0, 2)	(2, 0, 2)	(4, 0, 2)	(0, 0, 6)	(0, 0, 6)
s+10	s+6	(1, 0, 1)	(2, 0, 1)	(4, 0, 1)	(0, 0, 5)	(0, 0, 5)
s+11	s+7	(1, 0, 0)	(2, 0, 0)	(4, 0, 0)	(0, 0, 4)	(0, 0, 4)
s+12	s+8	(0, 0, 3)	(0, 0, 3)	(0, 0, 3)	(0, 0, 3)	(0, 0, 3)
s+13	s+9	(0, 0, 2)	(0, 0, 2)	(0, 0, 2)	(0, 0, 2)	(0, 0, 2)
s+14	s+10	(0, 0, 1)	(0, 0, 1)	(0, 0, 1)	(0, 0, 1)	(0, 0, 1)
s+15	s+11	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

3D Alignment Requirements

The vertical and horizontal alignment fields in the RENDER_SURFACE_STATE are ignored for standard tiling formats (TRMODE = NONE). In the case of standard tiling formats the alignment requirements are fixed and are provided for by the tables below for 3D (volumetric) surfaces.

Tile Mode	Bits per Element	Horizontal Alignment	Vertical Alignment	Depth Alignment
TileYS	128	16	16	16
	64	32	16	16
	32	32	32	16
	16	32	32	32
	8	64	32	32
TileYF	128	4	8	8
	64	8	8	8
	32	8	16	8

Tile Mode	Bits per Element	Horizontal Alignment	Vertical Alignment	Depth Alignment
	16	8	16	16
	8	16	16	16

Surface Padding Requirements

This section covers the requirements for padding around surfaces stored in memory, as there are cases where the device will overfetch beyond the bounds of the surface due to implementation of caches and other hardware structures.

Alignment Unit Size

This section documents the alignment parameters *i* and *j* to use, depending on the surface.

Alignment Parameters

Surface Defined By	Surface Format	Alignment Unit Width "*i*"	Alignment Unit Height "*j*"
3DSTATE_DEPTH_BUFFER	D16_UNORM	8	4
	not D16_UNORM	4	4
3DSTATE_STENCIL_BUFFER	N/A	8	8
SURFACE_STATE	BC*, ETC*, EAC*	4	4
	FXT1	8	4
	all others	set by **Surface Horizontal Alignment**	set by **Surface Vertical Alignment**

Surface Defined By	Surface Format	Alignment Unit Width "*i*"	Alignment Unit Height "*j*"
SURFACE_STATE	ASTC	Value of ASTC_2DBlockWidth (4, 5, 6, 8, 10, or 12)	ASTC_2DBlockHeight*4

Sampling Engine Surfaces

The sampling engine accesses texels outside of the surface if they are contained in the same cache line as texels that are within the surface. These texels will not participate in any calculation performed by the sampling engine and will not affect the result of any sampling engine operation, however if these texels lie outside of defined pages in the GTT, a GTT error will result when the cache line is accessed. In order to avoid these GTT errors, "padding" at the bottom and right side of a sampling engine surface is sometimes necessary.

It is possible that a cache line will straddle a page boundary if the base address or pitch is not aligned. All pages included in the cache lines that are part of the surface must map to valid GTT entries to avoid errors. To determine the necessary padding on the bottom and right side of the surface, refer to the table in Alignment Unit Size section for the i and j parameters for the surface format in use. The surface must then be extended to the next multiple of the alignment unit size in each dimension, and all texels contained in this extended surface must have valid GTT entries.

For example, suppose the surface size is 15 texels by 10 texels and the alignment parameters are i=4 and j=2. In this case, the extended surface would be 16 by 10. Note that these calculations are done in texels, and must be converted to bytes based on the surface format being used to determine whether additional pages need to be defined.

Buffer Padding Requirements

For compressed textures (BC*, FXT1, ETC*, and EAC* surface formats), padding at the bottom of the surface is to an even compressed row. This is equivalent to a multiple of 2*q*, where *q* is the compression block height in texels. Thus, for padding purposes, these surfaces behave as if $j = 2q$ only for surface padding purposes. The value of *j* is still equal to *q* for mip level alignment and QPitch calculation. For cube surfaces, an additional two rows of padding are required at the bottom of the surface. This must be ensured regardless of whether the surface is stored tiled or linear. This is due to the potential rotation of cache line orientation from memory to cache.

The above comments also apply to the ASTC* surface format.

For packed YUV, 96 bpt, 48 bpt, and 24 bpt surface formats, additional padding is required. These surfaces require an extra row plus 16 bytes of padding at the bottom in addition to the general padding requirements.

For linear surfaces, additional padding of 64 bytes is required at the bottom of the surface. This is in addition to the padding required above.

Programming Note	
Context:	Sampling Engine Surfaces.
For SURFTYPE_BUFFER, SURFTYPE_1D, and SURFTYPE_2D non-array, non-MSAA, non-mip-mapped surfaces in linear memory, the only padding requirement is to the next aligned 64-byte boundary beyond the end of the surface. The rest of the padding requirements documented above do not apply to these surfaces.	

Render Target and Media Surfaces

The data port accesses data (pixels) outside of the surface if they are contained in the same cache request as pixels that are within the surface. These pixels will not be returned by the requesting message, however if these pixels lie outside of defined pages in the GTT, a GTT error will result when the cache request is processed. In order to avoid these GTT errors, "padding" at the bottom of the surface is sometimes necessary.

Device2 PASID Capability Structures

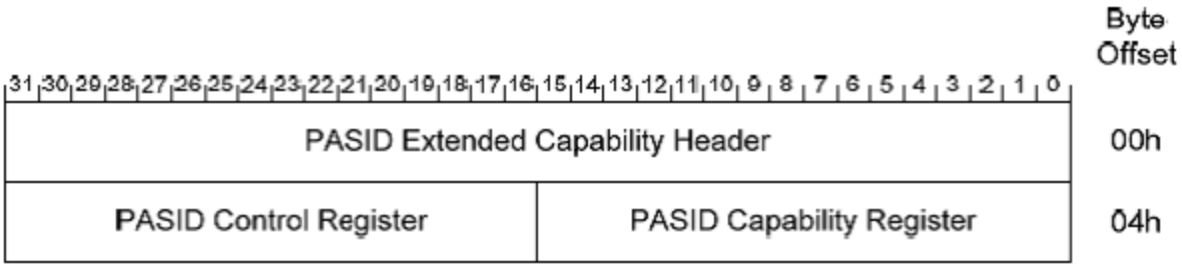
This part of the document describes the PCI-Express Extended Capability registers required on Processor Graphics (Device 2) device to enumerate Shared Virtual Memory (SVM) related capabilities.

SVM feature support on Device-2 is exposed through three distinct capability structures:

- **Process Address Space ID (PASID) Extended Capability.** PASID capability reports support for Process Address Space ID (PASID) on Device-2 compliant to PCI-Express PASID ECN.
- **Address Translation Services (ATS) Extended Capability.** ATS capability reports support for Device-TLBs on Device-2, compliant to PCI-Express ATS specification.
- **Page Request Extended Capability.** Page Request capability reports support for page-faults on Device-2, compliant to PCIExpress ATS 1.1 specification.

The following sections describe each of these capability structures and their implementation details.

PASID Extended Capability



Refer to PCI Express PASID ECN for more details.

Following sections describe the registers in the PASID Extended Capability structure.

PASID Extended Capability Header

Bits	Access	Default	Field
31:20	RO	Xh	**Next Capability Offset (NCO):** Offset to the next capability; Value 'X' in this field provides the offset for ATS Capability described in later sections
19:16	RO	1h	**Version (V):** Capability Version
15:0	RO	001Bh	**Capability ID (CAPID):** PASID Extended Capability ID.

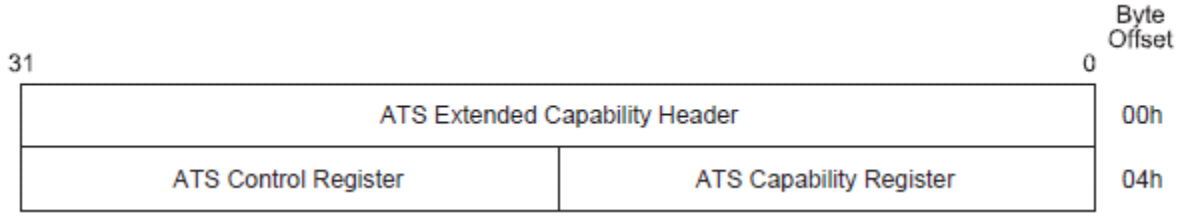
PASID Capability Register

Bits	Access	Default	Field
15:13	RO	0h	**Reserved.**
12:8	RO	14h	**Maximum PASID Width (MPW):** Indicates the width of the PASID field supported by the Endpoint. The value *n* indicates support for PASID values 0 through 2*n*-1 (inclusive). The value 0 indicates support for a single PASID (0). The value 20 indicates support for all PASID values (20 bits). This field must be between 0 and 20 (inclusive). Processor Graphics supports PASID width of 20-bits.
7:3	RO	0h	**Reserved.**
2	RO	1h	**Privilege Mode Supported (PMS):** If Set, the Endpoint supports operating in privileged and non-privileged modes, and requests-with-PASID can request privileged mode. If Clear, the Endpoint supports operating in non-privileged mode only, and will never request privileged mode in requests-with-PASID. On Processor Graphics, privileged and non-privileged mode are mapped to an attribute of the advanced context. Advanced contexts created for use by user-mode applications/drivers are considered non-privileged. Advanced context created for used by kernel mode software/drivers can be treated as privileged contexts. The privilege mode maps to the user/Supervisor (U/S) privilege checking in the first-level (IA-32e) paging. Advanced contexts supports a privileged/non-privileged attribute (part of context-state). Software is expected to create advanced-contexts with this attribute as non-privileged, for use by user-mode drivers/applications. For advanced contexts created for use by kernel mode software/drivers, this bit is configured can be privileged. Advanced-contexts configured as non-privileged function exactly the same as advanced contexts (i.e. restricted to user-mode virtual-addresses only). Advanced contexts configured as privileged can access both user-mode and kernel-mode virtual addresses, if Supervisor Request Enable field in extended-context-entry is Set.
1	RO	1h	**Execute Permission Supported (EPS)**: If Set, the Endpoint supports requests-with-PASID that request Execute permission. If Clear, the Endpoint will never request Execute permission for requests-with-PASID. On Processor Graphics, accesses by advanced contexts that fetch and execute graphics instructions are normally treated as requests that require execute permission. These maps to requests from GT Instruction Cache (IC) for advanced contexts. Advanced context accesses from Instruction Cache (IC) are checked for execute permission, if Execute Request Enable in extended-context-entry is Set.
0	RO	0h	**Reserved.**

PASID Control Register

Bits	Access	Default	Field
15:3	RO	0h	**Reserved.**
2	R/W	0b	**Privilege Mode Enable (PME):** When Set, the endpoint is permitted to request privileged mode in requests-with-PASID. If Clear, the Endpoint is not permitted to do so. Behavior is undefined if this bit changes value when ATS Enable field in ATS Capability is Set. If Privileged Mode Supported field in PASID Capability register is Clear, this field is treated as Reserved (0). Processor graphics does not use this field. Software is expected to Set this field before configuring extended-context-entry for Device-2 with Supervisor Request Enable field Set. For compatibility reasons, this field is implemented as RW.
1	R/W	0b	**Execute Permission Enable (EPE):** If Set, the Endpoint is permitted to request execute permission in requests-with-PASID. If Clear, the Endpoint is not permitted to do so. Behavior is undefined if this bit changes value when ATS Enable field in ATS Capability is Set. If Execute Permission Supported field in PASID Capability register is Clear, this field is treated as Reserved (0). Processor graphics does not use this field. Software is expected to Set this field before configuring extended-context-entry for Device-2 with Execute Request Enable field Set. For compatibility reasons, this field is implemented as RW.
0	R/W	0b	**PASID Enable (PE):** If Set, the Endpoint is permitted to generate requests-with-PASID. If Clear, the Endpoint is not permitted to do so. Behavior is undefined if this bit changes value when ATS Enable field in ATS Capability is Set. Processor Graphics does not use this field. Instead, all accesses from advanced contexts are treated as requested-with-PASID, subject to PASID-Enable field in the extended context-entry. Software is expected to Set this field before configuring extended-context entry for Device-2 with PASID Enable field Set. However, for compatibility reasons, this field is implemented as RW.

ATS Extended Capability



The following sections describe the registers in the ATS Extended Capability structure.

ATS Extended Capability Header

Bits	Access	Default	Field
31:20	RO	Xh	**Next Capability Offset (NCO).** Offset to the next capability; Value 'X' in this field provides the offset for ATS Capability described in later sections.
19:16	RO	1h	**Version (V).** Capability Version.
15:0	RO	000Fh	**Capability ID (CAPID).** PASID Extended Capability ID.

ATS Capability Register

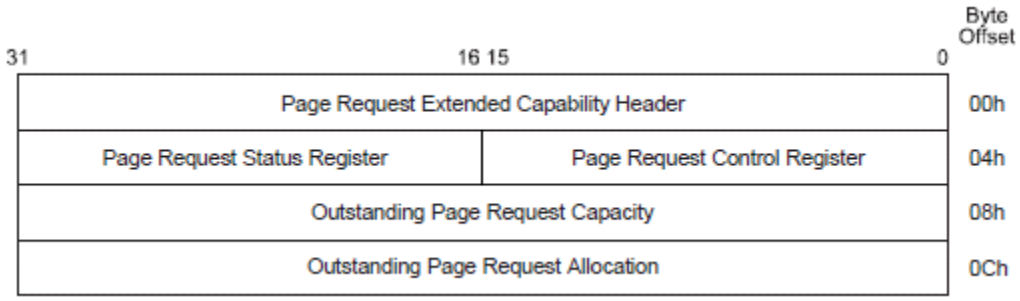
Bits	Access	Default	Field
15:6	RO	0h	Reserved.
5	RO	1b	**Page Aligned Request (PAR).** If Set, indicates the Untranslated Address is always aligned to a 4096 byte boundary. Processor graphics reports value of 1b.
4:0	RO	0h	**Invalidate Queue Depth (IQD).** The number of Invalidate Requests that the endpoint can accept before putting backpressure on the upstream connection. If 0h, the function can accept 32 Invalidate Requests. Processor Graphics does not use this field, and reports a value of 0h.

ATS Control Register

Bits	Access	Default	Field
15	R/W	0h	**ATS Enable (AE).** When Set, the function is enabled to cache translations. Processor graphics ignores this field, as GT uses GTLB as IOTLB, and only pretends to software that it has a Device-TLB. Software is expected to Set this field before configuring extended-context-entry for Device-2 with Page Request Enable field Set. For compatibility, this field is implemented as RW as software can read it to determine ATS enable status.
14:5	RO	0h	Reserved.

Bits	Access	Default	Field
4:0	RW	0h	**Smallest Translation Unit (STU).** This value indicates to the Endpoint the minimum number of 4096-byte blocks that is indicated in a Translation Completion or Invalidate Request. This is a power of 2 multiple and the number of blocks is 2 STU. A value of 0h indicates one block and value of 1F indicates 2 31 blocks. Processor graphics does not use this field, and reports a value of 0h indicating it uses 4KB as the smallest translation unit (smallest page-size).

Page Request Extended Capability



The following register descriptions define the requirement.

Page Request Extended Capability Header

Bits	Access	Default	Field
31:20	RO	0h	**Next Capability Offset (NCO):** Offset to the next capability; Value 'X' in this field provides the offset for ATS Capability described in later sections
19:16	RO	1h	**Version (V):** Capability Version
15:0	RO	0013h	**Capability ID (CAPID):** PASID Extended Capability ID.

Page Request Control Register

Bits	Access	Default	Field
15:2	RO	0h	**Reserved.**
1	RO	0	**Reset (RST):** When the Enable field is clear, or is being cleared in the same register update that sets this field, writing a 1b to this field, clears the associated implementation dependent page request credit counter and pending request state for the associated Page Request Interface. No action is initiated if this field is written to 0b or if this field is written with any value while the PRE field is Set. Read of this field return 0b. Processor graphics does not use this field, and implements it as read-only (0).
0	RW	0h	**Page Request Enable (PRE):** When Set, indicates that the page request interface on the endpoint is allowed to make page requests. If both this field and the Stopped field in Page Request Status register are Clear, then the page request interface will not issue new page requests, but has outstanding page requests for which page responses is not yet received. When this field transitions from 0 to 1, all the status fields in the Page-Request Status register are cleared. Enabling a page request interface that has not successfully stopped has indeterminate results. Processor graphics ignores this field, as GT uses GTLB as IOTLB, and only pretends to software that it has a Device-TLB. For compatibility, this field is implemented as RW as software can read it to determine Page- Request enable status. Software is expected to Set this field before configuring extended-context-entry for Device-2 with Page Request Enable field Set. Software is expected to respond to all page requests in the page-request queue before Clearing this field.

Page Request Status Register

Bits	Access	Default	Field
15:9	RO	0h	Reserved.
8	RO	0	**Stopped (S):** When this field is Set, the associated page request interface has stopped issuing additional page requests and that all previously issued Page Requests have completed. When this field is Clear the associate page request interface either has not stopped or has stopped issuing new Page Requests but has outstanding Page Requests. This field is only meaningful if Enable is Clear. If Enable is Set, this field is undefined. When the Enable field is Cleared, after having been previously Set, the interface transitions to the stopping state and Clears this field. After all page requests currently outstanding at the function have received responses, this field is Set and the interface enters the disabled state. If there were no outstanding page requests, this field may be Set immediately when Enable is Cleared. Resetting the interface causes an immediate transition to the disabled state. While in the stopping state, receipt of a Response Failure message results in the immediate transition to the disabled state (Setting this field). Processor Graphics has no direct use of this field. For compatibility reasons, this field is Set when Page-Request Enable (PRE) field in the Page-request Control register transitions from 1 to 0. When PRE transtions from 0 to 1, this field is Cleared.
7:2	RO	0h	Reserved.
1	RW1C	0h	**Unexpected Page Request Group Index (UPGRI):** When Set, indicates the function received a PRG response message containing a PRG index that has no matching request. A response failure. This field is Set by the Function and cleared when 1b is written to this field. Processor graphics Sets this field when it receives a *page_grp_resp_dsc* with PRG Index that does not match PRG index in any outstanding *page_grp_req_dsc*. Such a *page_grp_resp_dsc* is ignored. When Page- Request Enable (PRE) field in the Pagerequest Control register transtions from 0 to 1, this field is Cleared.
0	RW1C	0h	**Response Failure (RF):** When Set, indicates the function received a PRG response message indicating a response failure. The function expects no further response from the host (any received are ignored). This field is Set by the Function and cleared when 1b is written to this field. Processor graphics Sets this field when it receives a *page_grp_resp_dsc* or *page_stream_resp_dsc* with Response Code of Response Failure (1111b). The advanced context corresponding to the PASID in such response is terminated with error. When Page-Request Enable (PRE) field in the Page request Control register transitions from 0 to 1, this field is Cleared. HW writes 1 to this bit if it gets a failing page response.

Outstanding Page Request Capacity

Bits	Access	Default	Field
31:0	RO	8000h	**Outstanding Page Request Capacity (OPRC):** This register contains the number of outstanding page request messages the associated Page Request Interface physically supports. This is the upper limit on the number of pages that can be usefully allocated to the Page Request Interface. Processor Graphics device does not use this field.

Outstanding Page Request Allocation

Bits	Access	Default	Field
31:0	RO	0h	**Outstanding Page Request Allocation (OPRC):** This register contains the number of outstanding page request messages the associated Page Request Interface is allowed to issue (have outstanding at any given instance). Processor Graphics device does not use this field. Software is required to program this field with value $2^{(X+8)}$, where X is the value in Queue Size (QS) field in the Page Request Queue Address register. For compatibility reasons, this field is implemented as RW.

Atomics for Page Table Updates (MSQD)

The following atomics are added to GTI just for page table management and only needed by GAM to perform proper page walks:

The Following Atomics are only applicable in GTI and used for Page Walks. **Privilege and Instruction Access checks are not part of previous GPUs.** **R/W => Bit[0]** **Extended Access required => Bit[1]** **Write Protect Enable => Bit[2]** **Intermediate Entry => Bit[3]**					
Atomic Operation	**Opcode**	**Description**	**New Destination Value**	**Applicable**	**Return Value (Optional)**
Atomic_Page_update_0000	1100_0000	Read Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0001	1100_0001	Write Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0010	1100_0010	Read Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,10] if not set		new_dst

Atomic_Page_update_0011	1100_0011	Write Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_0100	1100_0100	Read Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0101	1100_0101	Write Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0110	1100_0110	Read Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0111	1100_0111	Write Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,6,10] if not set		new_dst

Atomic_Page_update_1000	1100_1000	Read Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5] if not set		new_dst
Atomic_Page_update_1001	1100_1001	Write Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst
Atomic_Page_update_1010	1100_1010	Read Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_1011	1100_1011	Write Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_1100	1100_1100	Read Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5] if not set		new_dst

Atomic_Page_update_1101	1100_1101	Write Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst
Atomic_Page_update_1110	1100_1110	Read Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_1111	1100_1111	Write Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst

Implementation

NOTE: Gen8 dropped the Execution privilege and Supervisor/User flag support. The following atomics are simplified for gen8 but need to be updated for gen9.

Atomic_Page_update_0000:

This paging walk is started on a READ access where the extended access bits in the context is disabled and write protection is disabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.

2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is "0".

Atomic_Page_update_0001:

This paging walk is started on a WRITE access where the extended access bits in the context is disabled and write protection is disabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) and D-bit (bit6) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is "0".

Atomic_Page_update_0010:

This paging walk is started on a READ access where the extended access bits in the context is enabled and write protection is disabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is "0".

Atomic_Page_update_0011:

This paging walk is started on a WRITE access where the extended access bits in the context is enabled and write protection is disabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) and EA-bit (bit10) and D-bit (bit6) set:
        i. If all set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is "0".

Atomic_Page_update_0100:

This paging walk is started on a READ access where the extended access bits in the context is disabled and write protection is enabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is "0".

Atomic_Page_update_0101:

This paging walk is started on a WRITE access where the extended access bits in the context is disabled and write protection is enabled. This is the final Page Table Entry.

ALU Decision:

1. If Page is present and Writes are allowed:
    - Is the A-bit (bit5) and D-bit (bit6) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page is present and Writes are not allowed:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
3. If Page is not present:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision:

- If bit[0] of the QW of interest is “0” => Page not present.
- If bit[1] of the QW of interest is “0” => Write permission fault.

Atomic_Page_update_0110:

This paging walk is started on a READ access where the extended access bits in the context is enabled and write protection is enabled. This is the final Page Table Entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

Atomic_Page_update_0111:

This paging walk is started on a WRITE access where the extended access bits in the context is enabled and write protection is enabled. This is the final Page Table Entry.

ALU Decision:

1. If Page is present and Writes are allowed:
    - Is the A-bit (bit5) and EA-bit (bit10) and D-bit (bit6) set:
        i. If all set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/EA/D-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page is present and Writes are not allowed::
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
3. If Page is not present:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0” or bit[1] of the QW of interest is “0”.

Atomic_Page_update_1000:

This paging walk is started on a READ access where the extended access bits in the context is disabled and write protection is disabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

Atomic_Page_update_1001:

This paging walk is started on a WRITE access where the extended access bits in the context is disabled and write protection is disabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault & 1GB/2MB Page:
    - Is the A-bit (bit5) and D-bit (bit6) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D-bit set, write to memory and return new-dest to GAM and end atomic.
2. If No-Page fault & Not 1GB/2MB Page:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
3. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

1GB/2MB Decision: It is a 1GB/2MB page if bit[7] of the intermediate walk entry is “1”.

Atomic_Page_update_1010:

This paging walk is started on a READ access where the extended access bits in the context is enabled and write protection is disabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

Atomic_Page_update_1011:

This paging walk is started on a WRITE access where the extended access bits in the context is enabled and write protection is disabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault & 1GB/2MB Page:
    - Is the A-bit (bit5) and EA-bit (bit10) and D-bit (bit6) set:
        i. If all set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If No-Page fault & Not 1GB/2MB Page:
    - Are the A-bit (bit5) and EA-bit (bit10) set:
        i. If both set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither not set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomi
3. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

1GB/2MB Decision: It is a 1GB/2MB page if bit[7] of the intermediate walk entry is “1”.

Atomic_Page_update_1100:

This paging walk is started on a READ access where the extended access bits in the context is disabled and write protection is enabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

Atomic_Page_update_1101:

This paging walk is started on a WRITE access where the extended access bits in the context is disabled and write protection is enabled. This is an intermediate page table entry.

ALU Decision:

1. If Page is present (1) & Writes are allowed (1) & 1GB/2MB Page (1):
    - Is the A-bit (bit5) and D-bit (bit6) set:
        i. If all set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D -bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page is present (1) and Writes are not allowed (0) & 1GB/2MB Page (1):
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
3. If Page is present (1) and Writes are allowed (1) & 1GB/2MB Page (0):
    - Is the A-bit (bit5) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
4. If Page is present (1) and Writes are not allowed (0) & 1GB/2MB Page (0):
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.
5. If Page is not present:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision:

- If bit[0] of the QW of interest is “0” => Page not present.
- If bit[1] of the QW of interest is “0” => Write permission fault.

1GB/2MB Decision: It is a 1GB/2MB page if bit[7] of the intermediate walk entry is “1”.

Atomic_Page_update_1110:

This paging walk is started on a READ access where the extended access bits in the context is enabled and write protection is enabled. This is an intermediate page table entry.

ALU Decision:

1. If No-Page fault:
    - Is the A-bit (bit 5)and EA-bit (bit10) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision: It is page fault, if bit[0] of the QW of interest is “0”.

Atomic_Page_update_1101:

This paging walk is started on a WRITE access where the extended access bits in the context is disabled and write protection is enabled. This is an intermediate page table entry.

ALU Decision:

1. If Page is present (1) & Writes are allowed (1) & 1GB/2MB Page (1):
    - Is the A-bit (bit5) and EA-bit (bit10) and D-bit (bit6) set:
        i. If all set, return the PTE to GAM and end-atomic (no writes).
        ii. If neither one of them set, modify the old-dest with A/D -bit set, write to memory and return new-dest to GAM and end atomic.
2. If Page is present (1) and Writes are not allowed (0) & 1GB/2MB Page (1):
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
3. If Page is present (1) and Writes are allowed (1) & 1GB/2MB Page (0):
    - Is the A-bit (bit5) and EA-bit (bit10) set:
        i. If set, return the PTE to GAM and end-atomic (no writes).
        ii. If not set, modify the old-dest with A/EA-bit set, write to memory and return new-dest to GAM and end atomic.
4. If Page is present (1) and Writes are not allowed (0) & 1GB/2MB Page (0):
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.
5. If Page is not present:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault Decision:

- If bit[0] of the QW of interest is “0” => Page not present.
- If bit[1] of the QW of interest is “0” => Write permission fault.

1GB/2MB Decision: It is a 1GB/2MB page if bit[7] of the intermediate walk entry is “1”.

Atomic_A_update_000:

Read access from a user context and a non-instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, just return the PTE to GAM and end-atomic.
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0", OR if bit[2]=0.

Implementation: The behavior should be exactly same as Atomic_A_update_010.

Atomic_A_update_001:

Read access from a user context and an instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, just return the PTE to GAM and end-atomic.
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0" OR bit[63]=1 OR if bit[2]=0.

Implementation: The behavior should be exactly same as Atomic_A_update_010.

Atomic_A_update_010:

Read access from a supervisor context and an non-instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, just return the PTE to GAM and end-atomic.
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is “0”.

Implementation: Identical to what is described here.

Atomic_A_update_011:

Read access from a supervisor context and an instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, just return the PTE to GAM and end-atomic.
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0" OR if bit[63]=1 OR if bit[2]=1.

Implementation: The behavior should be exactly same as Atomic_A_update_010.

Atomic_A_update_100:

Read access from a supervisor context and an instruction space and can be executed from supervisor or user pages.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, just return the PTE to GAM and end-atomic.
        ii. If not set, modify the old-dest with A-bit set, write to memory and return new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0" OR bit[63]=1.

Implementation: The behavior should be exactly same as Atomic_A_update_010.

Atomic_AD_update_101:

Write access from a user context and a non-instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, is the D-bit (bit6) set:
            A. If set, just return the PTE to GAM and end the atomic.
            B. If not set, modify the old-dest with D-bit set, write to memory and return new-dest to GAM and end atomic.
        ii. If not set, modify the old-dest with A and D-bit set, write to memory and return the new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0", OR if bit[2]=0 OR if bit[1]=0.

I**mplementation:** The behavior should be exactly same as Atomic_A_update_110.

Atomic_AD_update_110:

Write access from a supervisor context and an non-instruction space.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, is the D-bit (bit6) set:
            A. If set, just return the PTE to GAM and end the atomic.
            B. If not set, modify the old-dest with D-bit set, write to memory and return new-dest to GAM and end atomic
        ii. If not set, modify the old-dest with A and D-bit set, write to memory and return the new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is “0” OR bit[1]=0.

Implementation: Identical to what is described here.

Atomic_AD_update_111:

Write access from a supervisor context and a non-instruction space and RO is don't care.

1. If no-page fault:
    - Is the A-bit (bit5) set:
        i. If set, is the D-bit (bit6) set:
            A. If set, just return the PTE to GAM and end the atomic.
            B. If not set, modify the old-dest with D-bit set, write to memory and return new-dest to GAM and end atomic.
        ii. If not set, modify the old-dest with A and D-bit set, write to memory and return the new-dest to GAM and end atomic.
2. If page fault:
    - Return old-dest to GAM which will lead to GAM detecting the fault as well.

Page Fault decision: It is page fault, if bit[0] of the QW of interest is "0" OR bit[63]=1.

Implementation: The behavior should be to ignore bit[63] in the above proposed mechanism. The behavior should be exactly same as Atomic_A_update_110.

Atomic Operations between GPU and IA

IA cores are capable of doing atomic operations on any memory space defined as part of their page tables, MTRRs, and PAT. The core detects the memory type after applying all the checks, and if the end target is within WB space, it uses the optimized MESI protocol (i.e. RFO, WB) to complete the atomic operation. If the memory space is UC/WC/WT, it uses the bus lock to honor the atomic requirements.

GPUs are only capable of doing atomics via optimized MESI protocols; they do not have a mechanism to do bus locking. This restricts atomic operations to WB space if they happened to be between GPU and IA cores. We cannot guarantee the atomicity for UC/WC/WT.

There are multiple ways how GPU can handle atomics to non-WB space.

1. **Ignore the Memory Type:** GPU ignores the memory type for the atomic operation and performs it via RFO/WBMtoI as if the space is WB.

   If IA is relying on the LOCK for the same atomic, LOCK keeps the GPU off the bus for integrated solutions. And if GPU has the ownership of the line, a LOCKed read from IA still evicts the line from GT even if the IA has the line as UC/WT/WC in its memory space.
2. **Report as an error and fail the cycle:** GPU detects the mismatch and handles the cycle as error (i.e. write has no affect and read returns garbage) and reports the error as catastrophic to SW. App can be terminated by SW.
3. **Perform Bus Lock:** Similar to IA cores, GPU can support bus locks around atomic operations to UC/WT/WC spaces. However the initial implementations of shared atomic implementations do not have this option in GPU hardware.

Option#2 is the main mode where GPU is not expecting atomics to UC/WT/WC space and any such accesses are faulted as HW cannot guarantee the atomicity of the operation with the current implementation.

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 6: Command Stream Programming

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Graphics Command Formats 1
Command Header 2
Memory Interface Commands 5
2D Commands 7
3D Commands 8
VEBOX Commands 13
MFX Commands 13
Scheduling 15
RINGBUF — Ring Buffer Registers 15
Command Stream Virtual Memory Control 15
Execlists 15
Execlist Structure 16
Overall Context Layout 17
Context Layout 17
Ring Context 17
Ring Buffer 17
Context Descriptor Format 18
Logical Ring Context Format 18
Context Status 20
Render Engine Command Streamer (RCS) 23
Batch Buffer Privilege Register 23
Mode Registers 24
Logical Context Support 24
Context Save Registers 25
MI Commands for Render Engine 25
Watchdog Timer Registers 26
Interrupt Control Registers 27
Hardware-Detected Error Bit Definitions (for EIR EMR ESR) 28
Blitter Engine Command Streamer (BCS) 29
Logical Context Support 30
Mode Registers 31

MI Commands for Blitter Engine....32
Video Command Streamer (VCS)....33
Watchdog Timer Registers....34
Logical Context Support....34
Mode Registers....35
Registers in Media Engine....35
GFX Pending TLB Cycles Information Registers....35
Memory Interface Commands for Video Codec Engine....36
Video Enhancement Engine Command Interface....37
VECS_RINGBUF — Ring Buffer Registers....37
Logical Context Support....37
Mode Registers....38
MI Commands for Video Enhancement Engine....38
Watchdog Timers....39
The Per-Process Hardware Status Page....40
Preemption....41
Ring Buffer Scheduling....41
ExecList Scheduling....42
Command Streamer (CS) ALU Programming....43
MI Commands for Graphics Processing Engines....44
User Mode Privileged Commands....44

Graphics Command Formats

This section describes the general format of the graphics device commands.

Graphics commands are defined with various formats. The first DWord of all commands is called the *header* DWord. The header contains the only field common to all commands, the *client* field that determines the device unit that processes the command data. The Command Parser examines the client field of each command to condition the further processing of the command and route the command data accordingly.

Graphics commands vary in length, though are always multiples of DWords. The length of a command is either:

- Implied by the client/opcode
- Fixed by the client/opcode yet included in a header field (so the Command Parser explicitly knows how much data to copy/process)
- Variable, with a field in the header indicating the total length of the command

Note that command *sequences* require QWord alignment and padding to QWord length to be placed in Ring and Batch Buffers.

The following subsections provide a brief overview of the graphics commands by client type provides a diagram of the formats of the header DWords for all commands. Following that is a list of command mnemonics by client type.

Command Header

Render Command Header Format

Bits						
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**	
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count		
Reserved	001, 010	Opcode – 11111	23:19 Sub Opcode 00h – 01h		18:16 Reserved	15:0 DWord Count

TYPE	**31:29**	**28:27**	**26:24**	**23:16**	**15:8**	**7:0**
Common	011	00	Opcode – 000	Sub Opcode	Data	DWord Count
Common (NP)[1]	011	00	Opcode – 001	Sub Opcode	Data	DWord Count
Reserved	011	00	Opcode – 010 – 111			
Single Dword Command	011	01	Opcode – 000 – 001	Sub Opcode		N/A
Reserved	011	01	Opcode – 010 – 111			
Media State	011	10	Opcode – 000	Sub Opcode		Dword Count
Media Object	011	10	Opcode – 001 – 010	Sub Opcode	Dword Count	
Reserved	011	10	Opcode – 011 – 111			
3DState	011	11	Opcode – 000	Sub Opcode	Data	DWord Count
3DState (NP)[1]	011	11	Opcode – 001	Sub Opcode	Data	DWord Count
PIPE_Control	011	11	Opcode – 010		Data	DWord Count
3DPrimitive	011	11	Opcode – 011		Data	DWord Count
Reserved	011	11	Opcode – 100 – 111			
Reserved	100	XX				
Reserved	101	XX				
Reserved	110	XX				

Notes:

[1]The qualifier "NP" indicates that the state variable is non-pipelined and the render pipe is flushed before such a state variable is updated. The other state variables are pipelined (default).

Video Command Header Format

Bits						
TYPE	**31:29**	**28:24**		**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Reserved 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds			Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
TYPE	**31:29**	**28:27**	**26:24**	**23:16**		**15:0**
Reserved	011	00	XXX	XX		
MFX Single DW	011	01	000	Opcode: 0h		0
Reserved	011	01	1XX			
Reserved	011	10	0XX			
AVC State	011	10	100	Opcode: 0h – 4h		DWord Count
AVC Object	011	10	100	Opcode: 8h		DWord Count
VC1 State	011	10	101	Opcode: 0h – 4h		DWord Count
VC1 Object	011	10	101	Opcode: 8h		DWord Count
Reserved	011	10	11X			
Reserved	011	11	XXX			
TYPE	**31:29**	**28:27**	**26:24**	**23:21**	**20:16**	**15:0**
MFX Common	011	10	000	000	subopcode	DWord Count
Reserved	011	10	000	001-111	subopcode	DWord Count
AVC Common	011	10	001	000	subopcode	DWord Count
AVC Dec	011	10	001	001	subopcode	DWord Count
AVC Enc	011	10	001	010	subopcode	DWord Count
Reserved	011	10	001	011-111	subopcode	DWord Count
Reserved (for VC1 Common)	011	10	010	000	subopcode	DWord Count
VC1 Dec	011	10	010	001	subopcode	DWord Count
Reserved (for VC1 Enc)	011	10	010	010	subopcode	DWord Count
Reserved	011	10	010	011-111	subopcode	DWord Count
Reserved (MPEG2 Common)	011	10	011	000	subopcode	DWord Count
MPEG2 Dec	011	10	011	001	subopcode	DWord Count
Reserved (for MPEG2 Enc)	011	10	011	010	subopcode	DWord Count
Reserved	011	10	011	011-111	subopcode	DWord Count
Reserved	011	10	100-111	XXX		

Video Enhancement Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
Reserved	001, 010				

TYPE	31:29	28:27	26:24	23:21	20:16	15:12	11:0
VEBOX (Parallel Video Pipe)	011	10: Pipeline 00: Reserved 01: Reserved 11: Reserved	Command Opcode – 100	Sub Opcode A	Sub Opcode B	Reserved	Dword Count

Blitter Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
Reserved	001, 011				

TYPE	31:29	28:22	21:9	8:0
Blitter (2D)	010	Command Opcode	Command Dependent Data	Dword Count

Memory Interface Commands

Memory Interface (MI) commands are basically those commands which do not require processing by the 2D or 3D Rendering/Mapping engines. The functions performed by these commands include:

- Control of the command stream (e.g., Batch Buffer commands, breakpoints, ARB On/Off, etc.)
- Hardware synchronization (e.g., flush, wait-for-event)
- Software synchronization (e.g., Store DWORD, report head)
- Graphics buffer definition (e.g., Display buffer, Overlay buffer)
- Miscellaneous functions

All of the following commands are defined in *Memory Interface Commands*.

Memory Interface Commands for RCP

Opcode (28:23)	Command	Pipes
1 DWord		
00h	MI_NOOP	All
01h	MI_SET_PREDICATE	Render
02h	MI_USER_INTERRUPT	All
03h	MI_WAIT_FOR_EVENT	All
05h	MI_ARB_CHECK	All
06h	MI_RS_CONTROL	Render
07h	MI_REPORT_HEAD	All
08h	MI_ARB_ON_OFF	All except Blitter
09h	MI_URB_ATOMIC_ALLOC	Render
0Ah	MI_BATCH_BUFFER_END	All
0Bh	MI_SUSPEND_FLUSH	All
0Ch	MI_PREDICATE	Render
0Fh	MI_RS_CONTEXT	Render
2+ DWord		
10h	Reserved	
14h	MI_DISPLAY_FLIP	Render and Blitter
15h	Reserved	
16h	MI_SEMAPHORE_MBOX	All
17h	Reserved	
18h	MI_SET_CONTEXT	Render
1Ah	MI_MATH	All
1Bh	MI_SEMAPHORE_SIGNAL	All

Opcode (28:23)	Command	Pipes
1 DWord		
1Ch	MI_SEMAPHORE_WAIT	All
1Dh	MI_FORCE_WAKEUP	All except Render
1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	All
21h	MI_STORE_DATA_INDEX	All
22h	MI_LOAD_REGISTER_IMM	All
23h	MI_UPDATE_GTT	All
24h	MI_STORE_REGISTER_MEM	All
26h	MI_FLUSH_DW	All except Render
27h	MI_CLFLUSH	Render
29h	MI_LOAD_REGISTER_MEM	All
2Ah	MI_LOAD_REGISTER_REG	All
2Bh	MI_RS_STORE_DATA_IMM	Render
2Ch	MI_LOAD_URB_MEM	Render
2Dh	MI_STORE_URB_MEM	Render
2Eh	MI_MEM_TO_MEM	All
2Fh	MI_ATOMIC	All
Ring/Batch Buffer		
30h	Reserved	
31h	MI_BATCH_BUFFER_START	Render
32h-35h	Reserved	
36h	MI_CONDITIONAL_BATCH_BUFFER_END	All
37h-38h	Reserved	
39h-3Fh	Reserved	

2D Commands

The 2D commands include various flavors of BLT operations, along with commands to set up BLT engine state without actually performing a BLT. Most commands are of fixed length, though there are a few commands that include a variable amount of "inline" data at the end of the command.

All the following commands are defined in *Blitter Instructions*.

2D Command Map

Opcode (28:22)	Command
00h	Reserved
01h	XY_SETUP_BLT
02h	Reserved
03h	XY_SETUP_CLIP_BLT
04h-10h	Reserved
11h	XY_SETUP_MONO_PATTERN_SL_BLT
12h-23h	Reserved
24h	XY_PIXEL_BLT
25h	XY_SCANLINES_BLT
26h	XY_TEXT_BLT
27h-30h	Reserved
31h	XY_TEXT_IMMEDIATE_BLT
32h-3Fh	Reserved
40h	COLOR_BLT
41h	Reserved
42h	XY_FAST_COPY_BLT
43h	SRC_COPY_BLT
44h	Reserved
45h-4Fh	Reserved
50h	XY_COLOR_BLT
51h	XY_PAT_BLT
52h	XY_MONO_PAT_BLT
53h	XY_SRC_COPY_BLT
54h	XY_MONO_SRC_COPY_BLT
55h	XY_FULL_BLT
56h	XY_FULL_MONO_SRC_BLT
57h	XY_FULL_MONO_PATTERN_BLT
58h	XY_FULL_MONO_PATTERN_MONO_SRC_BLT
59h	XY_MONO_PAT_FIXED_BLT

Opcode (28:22)	Command
5Ah-70h	Reserved
71h	XY_MONO_SRC_COPY_IMMEDIATE_BLT
72h	XY_PAT_BLT_IMMEDIATE
73h	XY_SRC_COPY_CHROMA_BLT
74h	XY_FULL_IMMEDIATE_PATTERN_BLT
75h	XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT
76h	XY_PAT_CHROMA_BLT
77h	XY_PAT_CHROMA_BLT_IMMEDIATE
78h-7Fh	Reserved

3D Commands

The 3D commands are used to program the graphics pipelines for 3D operations.

Refer to the *3D* chapter for a description of the 3D state and primitive commands and the *Media* chapter for a description of the media-related state and object commands.

For all commands listed in **3D Command Map**, the Pipeline Type (bits 28:27) is 3h, indicating the 3D Pipeline.

3D Command Map

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	01h	Reserved	3D Pipeline
0h	02h	Reserved	3D Pipeline
0h	03h	Reserved	
0h	04h	3DSTATE_CLEAR_PARAMS	3D Pipeline
0h	05h	3DSTATE_DEPTH_BUFFER	3D Pipeline
0h	06h	3DSTATE_STENCIL_BUFFER	3D Pipeline
0h	07h	3DSTATE_HIER_DEPTH_BUFFER	3D Pipeline
0h	08h	3DSTATE_VERTEX_BUFFERS	Vertex Fetch
0h	09h	3DSTATE_VERTEX_ELEMENTS	Vertex Fetch
0h	0Ah	3DSTATE_INDEX_BUFFER	Vertex Fetch
0h	0Bh	3DSTATE_VF_STATISTICS	Vertex Fetch
0h	0Ch	3DSTATE_VF	Vertex Fetch
0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS	3D Pipeline
0h	0Eh	3DSTATE_CC_STATE_POINTERS	3D Pipeline
0h	10h	3DSTATE_VS	Vertex Shader
0h	11h	3DSTATE_GS	Geometry Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	12h	3DSTATE_CLIP	Clipper
0h	13h	3DSTATE_SF	Strips & Fans
0h	14h	3DSTATE_WM	Windower
0h	15h	3DSTATE_CONSTANT_VS	Vertex Shader
0h	16h	3DSTATE_CONSTANT_GS	Geometry Shader
0h	17h	3DSTATE_CONSTANT_PS	Windower
0h	18h	3DSTATE_SAMPLE_MASK	Windower
0h	19h	3DSTATE_CONSTANT_HS	Hull Shader
0h	1Ah	3DSTATE_CONSTANT_DS	Domain Shader
0h	1Bh	3DSTATE_HS	Hull Shader
0h	1Ch	3DSTATE_TE	Tesselator
0h	1Dh	3DSTATE_DS	Domain Shader
0h	1Eh	3DSTATE_STREAMOUT	HW Streamout
0h	1Fh	3DSTATE_SBE	Setup
0h	20h	3DSTATE_PS	Pixel Shader
0h	21h	3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP	Strips & Fans
0h	23h	3DSTATE_VIEWPORT_STATE_POINTERS_CC	Windower
0h	24h	3DSTATE_BLEND_STATE_POINTERS	Pixel Shader
0h	25h	3DSTATE_DEPTH_STENCIL_STATE_POINTERS	Pixel Shader
0h	26h	3DSTATE_BINDING_TABLE_POINTERS_VS	Vertex Shader
0h	27h	3DSTATE_BINDING_TABLE_POINTERS_HS	Hull Shader
0h	28h	3DSTATE_BINDING_TABLE_POINTERS_DS	Domain Shader
0h	29h	3DSTATE_BINDING_TABLE_POINTERS_GS	Geometry Shader
0h	2Ah	3DSTATE_BINDING_TABLE_POINTERS_PS	Pixel Shader
0h	2Bh	3DSTATE_SAMPLER_STATE_POINTERS_VS	Vertex Shader
0h	2Ch	3DSTATE_SAMPLER_STATE_POINTERS_HS	Hull Shader
0h	2Dh	3DSTATE_SAMPLER_STATE_POINTERS_DS	Domain Shader
0h	2Eh	3DSTATE_SAMPLER_STATE_POINTERS_GS	Geometry Shader
0h	2Fh	3DSTATE_SAMPLER_STATE_POINTERS_PS	Pixel Shader
0h	30h	3DSTATE_URB_VS	Vertex Shader
0h	31h	3DSTATE_URB_HS	Hull Shader
0h	32h	3DSTATE_URB_DS	Domain Shader
0h	33h	3DSTATE_URB_GS	Geometry Shader
0h	34h	3DSTATE_GATHER_CONSTANT_VS	Vertex Shader
0h	35h	3DSTATE_GATHER_CONSTANT_GS	Geometry Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	36h	3DSTATE_GATHER_CONSTANT_HS	Hull Shader
0h	37h	3DSTATE_GATHER_CONSTANT_DS	Domain Shader
0h	38h	3DSTATE_GATHER_CONSTANT_PS	Pixel Shader
0h	39h	3DSTATE_DX9_CONSTANTF_VS	Vertex Shader
0h	3Ah	3DSTATE_DX9_CONSTANTF_PS	Pixel Shader
0h	3Bh	3DSTATE_DX9_CONSTANTI_VS	Vertex Shader
0h	3Ch	3DSTATE_DX9_CONSTANTI_PS	Pixel Shader
0h	3Dh	3DSTATE_DX9_CONSTANTB_VS	Vertex Shader
0h	3Eh	3DSTATE_DX9_CONSTANTB_PS	Pixel Shader
0h	3Fh	3DSTATE_DX9_LOCAL_VALID_VS	Vertex Shader
0h	40h	3DSTATE_DX9_LOCAL_VALID_PS	Pixel Shader
0h	41h	3DSTATE_DX9_GENERATE_ACTIVE_VS	Vertex Shader
0h	42h	3DSTATE_DX9_GENERATE_ACTIVE_PS	Pixel Shader
0h	43h	3DSTATE_BINDING_TABLE_EDIT_VS	Vertex Shader
0h	44h	3DSTATE_BINDING_TABLE_EDIT_GS	Geometry Shader
0h	45h	3DSTATE_BINDING_TABLE_EDIT_HS	Hull Shader
0h	46h	3DSTATE_BINDING_TABLE_EDIT_DS	Domain Shader
0h	47h	3DSTATE_BINDING_TABLE_EDIT_PS	Pixel Shader
0h	48h	3DSTATE_VF_HASHING	Vertex Fetch
0h	49h	3DSTATE_VF_INSTANCING	Vertex Fetch
0h	4Ah	3DSTATE_VF_SGVS	Vertex Fetch
0h	4Bh	3DSTATE_VF_TOPOLOGY	Vertex Fetch
0h	4Ch	3DSTATE_WM_CHROMA_KEY	Windower
0h	4Dh	3DSTATE_PS_BLEND	Windower
0h	4Eh	3DSTATE_WM_DEPTH_STENCIL	Windower
0h	4Fh	3DSTATE_PS_EXTRA	Windower
0h	50h	3DSTATE_RASTER	Strips & Fans
0h	51h	3DSTATE_SBE_SWIZ	Strips & Fans
0h	52h	3DSTATE_WM_HZ_OP	Windower
0h	53h	3DSTATE_INT (internally generated state)	3D Pipeline
0h	54h	3DSTATE_RS_CONSTANT_POINTER	Resource Streamer
0h	55h	3DSTATE_VF_COMPONENT_PACKING	Vertex Fetch
0h	58h	Reserved	
0h	59h	Reserved	
0h	5Ah	Reserved	

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	5Bh	Reserved	
0h	5Ch	Reserved	
0h	5Dh-69h	Reserved	
0h	56h	Reserved	
0h	57h-59h	Reserved	
0h	60h-68h	Reserved	
0h	69h	Reserved	
0h	6Ah-6Bh	Reserved	
0h	6Ch	Reserved	
0h	6Dh	Reserved	
0h	6Eh-FFh	Reserved	
1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0	Sampling Engine
1h	03h	Reserved	
1h	04h	3DSTATE_CHROMA_KEY	Sampling Engine
1h	05h	Reserved	
1h	06h	3DSTATE_POLY_STIPPLE_OFFSET	Windower
1h	07h	3DSTATE_POLY_STIPPLE_PATTERN	Windower
1h	08h	3DSTATE_LINE_STIPPLE	Windower
1h	0Ah	3DSTATE_AA_LINE_PARAMS	Windower
1h	0Bh	3DSTATE_GS_SVB_INDEX	Geometry Shader
1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1	Sampling Engine
1h	0Dh	3DSTATE_MULTISAMPLE	Windower
1h	0Eh	3DSTATE_STENCIL_BUFFER	Windower
1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER	Windower
1h	10h	3DSTATE_CLEAR_PARAMS	Windower
1h	11h	3DSTATE_MONOFILTER_SIZE	Sampling Engine
1h	12h	3DSTATE_PUSH_CONSTANT_ALLOC_VS	Vertex Shader
1h	13h	3DSTATE_PUSH_CONSTANT_ALLOC_HS	Hull Shader
1h	14h	3DSTATE_PUSH_CONSTANT_ALLOC_DS	Domain Shader
1h	15h	3DSTATE_PUSH_CONSTANT_ALLOC_GS	Geometry Shader
1h	16h	3DSTATE_PUSH_CONSTANT_ALLOC_PS	Pixel Shader
1h	17h	3DSTATE_SO_DECL_LIST	HW Streamout
1h	18h	3DSTATE_SO_BUFFER	HW Streamout
1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC	Resource Streamer

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
1h	1Ah	3DSTATE_GATHER_POOL_ALLOC	Resource Streamer
1h	1Bh	3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC	Resource Streamer
1h	1Ch	3DSTATE_SAMPLE_PATTERN	Windower
1h	1Dh	3DSTATE_URB_CLEAR	3D Pipeline
1h	1Eh-FFh	Reserved	
2h	00h	PIPE_CONTROL	3D Pipeline
2h	01h-FFh	Reserved	
3h	00h	3DPRIMITIVE	Vertex Fetch
3h	01h-FFh	Reserved	
4h-7h	00h-FFh	Reserved	

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
Common (pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	0h	03h	STATE_PREFETCH	Graphics Processing Engine
0h	0h	04h-FFh	Reserved	
Common (non-pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	1h	00h	Reserved	N/A
0h	1h	01h	STATE_BASE_ADDRESS	Graphics Processing Engine
0h	1h	02h	STATE_SIP	Graphics Processing Engine
0h	1h	03h	Reserved	3D Pipeline
0h	1h	04h	GPGPU CSR BASE ADDRESS	Graphics Processing Engine
0h	1h	05h-1Dh	Reserved	
0h	1h	1Eh	Reserved	
0h	1h	1Fh-20h	Reserved	
0h	1h	21h-24h	Reserved	
0h	1h	25h-FFh	Reserved	N/A
Reserved	**Bits 26:24**	**Bits 23:16**		
0h	2h-7h	XX	Reserved	N/A

VEBOX Commands

The VEBOX commands are used to program the Video Enhancement engine attached to the Video Enhancement Command Parser.

VEBOX Command Map

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command
2h	4h	0h	0h	VEBOX_SURFACE_STATE
2h	4h	0h	2h	VEBOX_STATE
2h	4h	0h	3h	VEBOX_DI_IECP
2h	4h	0h	1h	VEBOX_TILING_CONVERT

MFX Commands

The MFX (MFD for decode and MFC for encode) commands are used to program the multi-format codec engine attached to the Video Codec Command Parser. See the *MFD* and *MFC* chapters for a description of these commands.

MFX state commands support direct state model and indirect state model. Recommended usage of indirect state model is provided here (as a software usage guideline).

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
MFX Common (State)							
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	N/A
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	N/A
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	N/A
2h	0h	0h	7-8h	Reserved	N/A	N/A	N/A
MFX Common (Object)							
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	N/A	Yes
2h	0h	0h	4-1Fh	Reserved	N/A	N/A	N/A
AVC Common (State)							
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	N/A
2h	1h	0h	1h	MFX_AVC_QM_STATE	MFX	IMAGE	N/A
2h	1h	0h	2h	MFX_AVC_DIRECTMODE_STATE	MFX	SLICE	N/A
2h	1h	0h	3h	MFX_AVC_SLICE_STATE	MFX	SLICE	N/A

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	1h	0h	4h	MFX_AVC_REF_IDX_STATE	MFX	SLICE	N/A
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	N/A
2h	1h	0h	6-1Fh	Reserved	N/A	N/A	N/A
AVC Dec							
2h	1h	1h	0-7h	Reserved	N/A	N/A	N/A
2h	1h	1h	8h	MFD_AVC_BSD_OBJECT	MFX	N/A	No
2h	1h	1h	9-1Fh	Reserved	N/A	N/A	N/A
AVC Enc							
2h	1h	2h	0-1h	Reserved	N/A	N/A	N/A
2h	1h	2h	2h	MFC_AVC_FQM_STATE	MFX	IMAGE	N/A
2h	1h	2h	3-7h	Reserved	N/A	N/A	N/A
2h	1h	2h	8h	MFC_AVC_PAK_INSERT_OBJECT	MFX	N/A	N/A
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	N/A	Yes
2h	1h	2h	A-1Fh	Reserved	N/A	N/A	N/A
2h	1h	2h	0-1Fh	Reserved	N/A	N/A	N/A
VC1 Common							
2h	2h	0h	0h	MFX_VC1_PIC_STATE	MFX	IMAGE	N/A
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	N/A
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	N/A
2h	2h	0h	2-1Fh	Reserved	N/A	N/A	N/A
VC1 Dec							
2h	2h	1h	0-7h	Reserved	N/A	N/A	N/A
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	N/A	Yes
2h	2h	1h	9-1Fh	Reserved	N/A	N/A	N/A
VC1 Enc							
2h	2h	2h	0-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Common							
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	N/A
2h	3h	0h	1h	MFX_MPEG2_QM_STATE	MFX	IMAGE	N/A
2h	3h	0h	2-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Dec							
2h	3h	1h	1-7h	Reserved	N/A	N/A	N/A
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	N/A	Yes
2h	3h	1h	9-1Fh	Reserved	N/A	N/A	N/A

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
MPEG2 Enc							
2h	3h	2h	0-1Fh	Reserved	N/A	N/A	N/A
The Rest							
2h	4-5h, 7h	x	x	Reserved	N/A	N/A	N/A

Scheduling

RINGBUF — Ring Buffer Registers

See the "Device Programming Environment" chapter for detailed information on these registers.

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Command Stream Virtual Memory Control

Per-Process GTT (PPGTT) is setup for an engine (Render, Blitter, Video and Video Enhancement) by programming corresponding Page Directory Pointer (PDP) registers listed below. Refer "Graphics Translation Tables" in "Memory Overview" for more details on Per-Process page table entries and related translations.

Execlists

Execlists are the method by which new contexts are submitted for execution. Note that this mechanism cannot be used when the **Execlist Enable** bit in the corresponding engines MODE register is not set, i.e GFX_MODE register for Render Engine, BLT_MODE register for Blitter Engine, VCS_MODE register for Video Engine, or VECS_MODE register for Video Enhancement Engine. If this bit is not set in the engine's MODE register, writing to the registers in this section is UNDEFINED.

Broadwell implements two execlists. Each execlist can have up to two context descriptors in it, each describing a context to run. SW assembles an execlist by writing each of the context descriptor elements to the Execlist Submit Port register. Writing the final DWord triggers the submission. It is the responsibility of SW to keep track of when an empty execlist entry is available to receive a new execlist submitted via the Submit Port. Submitting a new execlist when there is already a pending execlist (in addition to the current execlist) is UNDEFINED. In general, the interrupt indicating that the pending execlist has become the current execlist should always be observed before a new pending execlist is

submitted. This includes the case where the ring is idle and the very first execlist is submitted; it should not be assumed that this execlist becomes the current list instantaneously.

The submission of a new execlist (known as a preemption request) is interpreted as a request to switch execlists as soon as possible. This is the only trigger for an execlist switch. Within an execlist, a switch from one element (context) to the next can be triggered for several reasons, all of which are synchronous to what the running context itself is doing. Once a context is switched out, the relevant context state and context descriptor doesn't exist in HW, only way the context can be brought back in to HW is by SW resubmitting the context through Execlist Submit Port.

SW must ensure the contexts submitted to both the context descriptors in the execlist are different; i.e SW must not submit the same context descriptor to both the elements of the execlist.

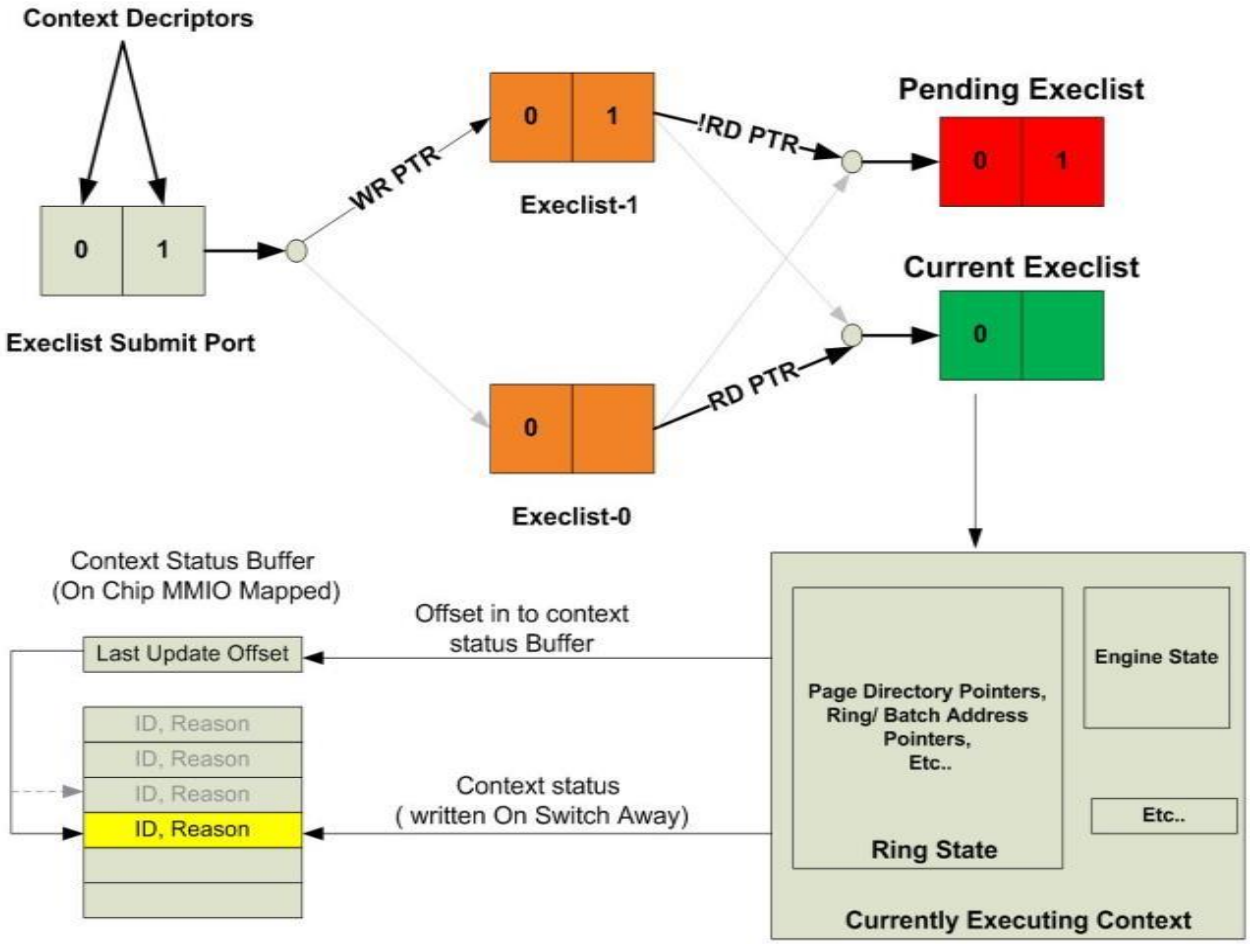
The following are Execlist Registers:

Execlist Submit Port Register

Execlist 1 Contents

EXECLIST_STATUS - Execlist Status Register

Execlist Structure



Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Context Descriptor Format Structure

Overall Context Layout

Context Layout

When Execlists are enabled, the Context Image for the rendering engine consists of 20 4K pages:

Per-Process HW Status Page (4K)
Register State Context

When Execlists are disabled, the context image doesn't consist the Per-Process HW status page.

Register State context is explained in detail in "Register State Context" Section.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size. Note that the last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in Execution List mode of scheduling. Ring Buffer registers are directly programmed in Ring Buffer mode of scheduling.

Context Descriptor Format

Context Descriptor Format

Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Programming Note on Context ID field in the Context Descriptor

This section describes the current usage by SW.

Layout:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Eng. ID				SW Counter					HW Use		SW Context ID																					

Logical Ring Context Format

Context descriptor has the graphics virtual address pointing to the logical context in memory. Logical context has all the details required for an engine to execute a context. This is the only means through which software can pass on all the required information to hardware for executing a context. Engine first step on selecting a context for execution is to restore (fetch-context save) the logical context from memory to setup the appropriate state in the hardware. Engine on switching out the context from execution saves (store- context restore) the latest updated state to logical context in memory, the updated state is result of the command buffer execution.

When execution lists are enabled, the Logical Context of each engine (Render, Video, Blitter, Video Enhancement, etc.) primarily consists of the following sections:

- Per-Process HW Status Page (4K)
- Ring Context (Ring Buffer Control Registers, Page Directory Pointers, etc.)
- Engine Context (PipelineState, Non-pipelineState, Statistics, MMIO)

Per-Process of HW status Page (PPHWSP)

This is a 4KB scratch space memory allocated for each of the context in global address space. First few cachelines are used by the engine for implicit reports like auto-report of head pointer, timestamp statistics associated with a context execution, rest of the space is available for software as scratch space for reporting fences through MI commands. Context descriptor points to the base of Per-Process HW status page. See the PPHWSP format in **PPHWSP_LAYOUT**.

Logical Ring Context

Logical Ring Context starts immediately following the PPHWSP in memory. Logical ring context is five cachelines in size. This is the minimal set of hardware state required to be programmed by SW for setting up memory access and the ring buffer for a context to be executed on an engine. Memory setup is required for appropriate address translation in the memory interface. Ring buffer details the location of the ring buffer in global graphics virtual address space with its corresponding head pointer and the tail pointer. Ring context also has "Context Save/Restore Control Register-CTXT_SR_CTL" which details the engine context save/restore format. Engine first restores the Logical Ring Context and upon processing CTXT_SR_CTL it further decides the due course of Engine Context restore. Logical Ring Context is mostly identical across all engines. Logical ring context is saved to memory with the latest up to date state when a context is switched out.

Engine Context

Engine context starts immediately following the logical ring context in memory. This state is very specific to an engine and differs from engine to engine. This part of the context consists of the state from all the units in the engine that needs to be save/restored across context switches. Engine restores the engine context following the logical ring context restore. It is tedious for software to populate the engine context as per the requirements, it is recommended to implicitly use engine to populate this portion of the context. Below method can be followed to achieve the same:

- When a context is submitted for the first time for execution, SW can inhibit engine from restoring engine context by setting the "Engine Context Restore Inhibit" bit in CTXT_SR_CTL register of the logical ring context. This will avoid software from populating the Engine Context. Software must program all the state required to initialize the engine in the ring buffer which would initialize the hardware state. On a subsequent context save engine will populate the engine context with appropriate values.
- Above method can be used to create a complete logical context with engine context populated by the hardware. This Logical context can be used as an Golden Context Image or template for subsequently created contexts.

Engine saves the engine context following the logical ring context on switching out a context.

The detailed format of the logical ring context (Blitter/Video/VideoEnhancement) is documented in the Memory Data Formats chapter.

The detailed formats of the Render Logical Ring and Engine Context, including their size, is mentioned in the "Register/State Context" topic for each product.

Context Status

A context switch interrupt will be sent anytime a context switch or execlist change occurs (including the execlist change without context switch scenario described in the ELSP -- Execlist Submit Port Register section). A status QW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. A copy of the Context Status Buffer is also maintained ON CHIP inside the command streamer, which is MMIO mapped and can be read/written using MMIO access.

Context Status Buffer in Global Hardware Status Page is exercised when IA based scheduling is done. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) execlist is submitted or when HW is idle, the Context ID in the first Context Status Qword will be UNDEFINED, this is indicated by setting IDLE to ACTIVE bit in the context status.

Format of Context Status QWord

Bits	Description
63:32	**Context ID**
31:29	Reserved
28	Reserved
27:24	Reserved
23:20	Reserved
19:16	**Display Plane.** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set. 0000b: Reserved (Look at field 14:12) 0001b: Reserved 0010b: Reserved 0011b: Display Plane-7 0100b: Display Plane-8 0101b: Display Plane-9 0110b: Display Plane-10 0111b: Display Plane-11 1000b: Display Plane-12 1001b to 1111b: Reserved
15	**Lite Restore.** This bit is only valid only when Preempted bit is set. When set, this bit indicates a given context got preempted with the same context resulting in Lite Restore in HW.

Bits	Description
14:12	**Display Plane.** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field must be looked at only when Display Plane on bits [19:16] is "0000b" and when one of the "Wait on Scanline" or "Wait on Vblank" or "Wait on sync Flip" is set. 000b: Display Plane-1(Pipe A if Wait on V-blank) 001b: Display Plane-2(Pipe B if Wait on V-blank) 010b: Display Plane-3(Pipe C if Wait on V-blank) 011b: Display Plane-4 100b: Display Plane-5 101b: Display Plane-6
11	**Semaphore Wait Mode** 0: Signal Mode 1: Poll Mode This field is valid and must be looked at only when the "Wait on Semaphore" field is set.
10	Reserved
9	Reserved
8	**Wait on Scanline** has resulted in context switch.
7	**Wait on Semaphore** has resulted in context switch.
6	**Wait on V-Blank** has resulted in context switch.
5	**Wait on Sync Flip** has resulted in context switch.
4	**Context Complete** Element is completely processed (Head eqv to Tail) and resulted in a context switch.
3	**ACTIVE to IDLE** following this context switch there is no active element available in HW to execute.
2	**Element Switch.** Context Switch happened from first element in the current execlist to the second element of the same execlist.
1	**Preempted.** Submission of a new execlist has resulted in context switch. The switch is from element in current execlist to element in pending execlist.
0	**IDLE to ACTIVE.** Execlist Submitted when HW is IDLE. When this bit is set rest of the fields in CSQ are not valid.

Context Status should be inferred as described in the tables below. In the two tables below only one of the context switch types will be set and it's quite possible multiple context switch reasons are set. A "Y" in a cell indicates the possibility of the context switch type for the corresponding context switch reason.

Inference of Context Status

Ctx Switch Type Ctx Switch Reason	IDLE to Active	Preempted/ Execlist Switch	**Element Switch	ACTIVE to IDLE
Context Complete	X	Y	Y	Y
Wait on Sync Flip	X	Y	Y	Y
Wait on V-Blank	X	Y	Y	Y
Wait on ScanLine	X	Y	Y	Y
Wait on Semaphore	X	Y	Y	Y

Ctx Switch Type Ctx Switch Reason	IDLE to Active	Preempted/ Execlist Switch	**Element Switch	ACTIVE to IDLE
High Priority Context	Y	Y	X	Y

** This field is not valid when High Priority Context field is set and HW must force it to '0'.

When SW services a context switch interrupt, it should read the Context Status Buffer beginning where it left off reading the last time it serviced a context switch interrupt. It should read up to the **Context Status Buffer Write Pointer,** which is recorded in the Context Status Buffer Pointer register. At the end of the context switch interrupt processing SW will update the **Context Status Buffer Read Pointer** with the write buffer pointer value. The status QWs can be examined to determine which contexts were switched out between context interrupt service intervals, and why.

Number of Context Status Entries

Number of Status Entries
6 (QW) Entries

Status QWords are written to the Context Status Buffer at incrementing locations. The Context Status Buffer has a limited size (see Table Number of Context Status Entries) and simply wraps around to the beginning when the end is reached. Normally the number of status updates that can occur without SW intervening to submit a new execlist (and presumably reading any new status) is the number of execlists times the maximum number of context elements per execlist. Also note that there is no predictable relationship between a context's position in an execlist and the position of its corresponding status QWord in the Context Status Buffer.

The Context Status Buffer fits into a single cacheline so that the whole buffer is read from memory at once if the driver performs a cacheable read.

Format of the Context Status Buffer

QW	Description
7	**Last Written Status Offset.** The lower byte of this QWord is written on every context switch with the (pre-increment) value of the b>Context Status Buffer Write Pointer. The lower 3 bits increment for every status QWord write; bits[7:3] are reserved and must be '0'. The lowest 3 bits indicate which of the Context Status QWords was just written. The rest of the bits [63:8] are reserved.
6	Reserved: MBZ
5:0	**Context Status QWords.** A circular buffer of context status QWs. As each context is switched away from, its status is written here at ascending QWs as indicated by the **Last Written Status Offset**. Once QW 5 has been written, the pointer wraps around so that the next status will be written at QW0. Format = ContextStatusDW

The following are Context Status Registers:

CTXT_ST_BUF - Context Status Buffer Contents

Render Engine Command Streamer (RCS)

The RCS (Render Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Render Engine. It is responsible for fetching, decoding, and dispatching of data packets (3D/Media Commands with the header DWord removed) to the front end interface module of Render Engine.

Logic Functions Included
• MMIO register programming interface. • DMA action for fetching of ring data from memory. • Management of the Head pointer for the Ring Buffer. • Decode of ring data and sending it to the appropriate destination: 3D (Vertex Fetch Unit) & GPGPU. • Handling of user interrupts. • Flushing the 3D and GPGPU Engine. • Handle NOP.
• DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

The register programming bus is a DWord interface bus that is driven by the configuration master. The RCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x2000 to 0x27FF. The Gx and MFX Engines use semaphore to synchronize their operations.

RCS operates completely independent of the MFx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside RCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (8 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards Vertex Fetch Unit or GPPGU engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Batch Buffer Privilege Register

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

Mode Registers

The following are the Mode Registers:

Register
INSTPM - Instruction Parser Mode Register
EXCC - Execute Condition Code Register
NOPID - NOP Identification Register
CSPREEMPT - CSPREEMPT
RS_PRE_HINT - RS Preemption Hint
RS_PREEMPTION_HINT_UDW - RS Preemption Hint UDW
IDLEDLY - Idle Switch Delay
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

Logical Context Support

The following are the Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
CCID - Current Context Register
CXT_SIZE - Context Sizes
RS_CXT_OFFSET - Resource Streamer Context Offset
URB_CXT_OFFSET - URB Context Offset
VF_CXT_OFFSET - Vertex Fetch Context Offset
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
SBB_STATE - Second Level Batch Buffer State Register
PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0
PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1
PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2
PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0
PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1
PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
R_PWR_CLK_STATE - Render Power Clock State Register
CS_CONTEXT_STATUS1 - Context Status1 for RCS-BE

Context Save Registers

The following are the Context Save Registers:

Register
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
BB_OFFSET - Batch Offset Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
GAM_CTX - GAM Context Save Register
TDL_CONTEXT_SAVE - Context Save Request to TDL

MI Commands for Render Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the original graphics processing engine. The term "for Rendering Engine" in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

Commands
MI_NOOP
MI_ARB_CHECK
MI_ARB_ON_OFF
MI_BATCH_BUFFER_START
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_CLFLUSH
MI_MATH

Commands
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_FORCE_WAKEUP

Watchdog Timer Registers

These registers together implement a watchdog timer. Writing ones to the control register enables the counter, and writing zeros disables the counter. The second register is programmed with a threshold value which, when reached, signals an interrupt that then resets the counter to 0. Program the threshold value before enabling the counter or extremely frequent interrupts may result.

Note: The counter itself is not observable. It increments with the main render clock.

Programming Notes: When watch dog timer is enabled, HW does not trigger any kind of idle sequences. SW must enable and disable watch dog timer for any given workload within the same command buffer dispatch. SW must disable watch dog timer around semaphore waits and wait for events commands so that HW can trigger appropriate idle sequence for power savings.

Interrupt Control Registers

The Interrupt Control Registers described in this section all share the same bit definition. The bit definition is as follows:

Bit Defintion for Interrupt Control Registers:

Bit Definition for Interrupt Control Registers - Render

Bit Definition for Interrupt Control Registers - Blitter

Bit Definition for Interrupt Control Registers - Media#1

Bit Definition for Interrupt Control Registers - Media#2

Bit Definition for Interrupt Control Registers - Video Enhancement

The following table specifies the settings of interrupt bits stored upon a "Hardware Status Write" due to ISR changes:

Bit	Interrupt Bit	ISR Bit Reporting Via Hardware Status Write (When Unmasked Via HWSTAM)
9	Reserved	
8	**Context Switch Interrupt.** Set when a context switch has just occurred.	Not supported to be unmasked.
7	**Page Fault.** This bit is set whenever there is a pending PPGTT (page or directory) fault. This interrupt is for handling Legacy Page Fault interface for all Command Streamers (BCS, RCS, VCS, VECS). When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c "Page Fault Support" section for more details.	Set when event occurs, cleared when event cleared. Not supported to be unmasked.
6	**Media Decode Pipeline Counter Exceeded Notify Interrupt.** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery.	Not supported to be unmasked. Only for Media Pipe.
5	**L3 Parity interrupt**	Only for Render Pipe
4	**Flush Notify Enable**	0
3	**Master Error**	Set when error occurs, cleared when error cleared.
2	Reserved	
0	**User Interrupt**	0

IMR - Interrupt Mask Register
Command Streamer (All) > Hardware Status Mask Register

Hardware-Detected Error Bit Definitions (for EIR EMR ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR, and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following structures describe the Hardware-Detected Error bits:

The following structures describe the Hardware-Detected Error bits:

Error Bits
RCS Hardware-Detected Error Bit Definitions Structure
BCS Hardware-Detected Error Bit Definitions Structure
VCS Hardware-Detected Error Bit Definitions Structure
VECS Hardware-Detected Error Bit Definitions Structure

The following are the EIR, EMR and ESR registers:

Registes
EIR - Error Identity Register
EMR - Error Mask Register
ESR - Error Status Register

Blitter Engine Command Streamer (BCS)

The BCS (Blitter Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Blitter Engine. It is responsible for fetching, decoding, and dispatching of data packets (Blitter Commands) to the front end interface module of Blitter Engine.

Logic Functions Included
• MMIO register programming interface. • DMA action for fetching of ring data from memory. • Management of the Head pointer for the Ring Buffer. • Decode of ring data and sending it to the blit engine. • Handling of user interrupts. • Flushing the Blitter Engine. • Handle NOP.
• DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

The register programming bus is a DWord interface bus that is driven by the configuration master. The BCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x22000 to 0x224FF. The Blitter, Render and Media Engines use semaphore to synchronize their operations.

BCS operates completely independent of the other render and media command streams.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside BCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (8 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards Blit Engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Logical Context Support

The following are the Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Mode Registers

The following table describes the Mode Registers.

Registers
BCS_CXT_SIZE - BCS Context Sizes
MI_MODE - Mode Register for Software Interface

Mode Registers (continued)

Reisters
INSTPM - Instruction Parser Mode Register
EXCC - Execute Condition Code Register
IDLEDLY - Idle Switch Delay
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait
RESET_CTRL - Reset Control Register
PREEMPTION_HINT - Preemption Hint
PREEMPTION_HINT_UDW - Preemption Hint Upper DWord

Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

MI Commands for Blitter Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the blitter graphics processing engine. The term "for Blitter Engine" in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine and the Rendering Engine.

The commands detailed in this chapter are used across products. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

Commands
MI_NOOP
MI_ARB_CHECK
MI_ARB_ON_OFF
MI_BATCH_BUFFER_START

The following table lists the non-privileged registers that can be written to from a non-secure batch buffer executed from Render Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	22600h	32
BCS_SWCTRL	22200h	32

Commands
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_FLUSH_DW
MI_MATH
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM

Commands
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_FORCE_WAKEUP

Video Command Streamer (VCS)

The VCS (Video Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the MFD Engine. It is responsible for fetching, decoding, and dispatching of data packets (Media Commands with the header DWord removed) to the front end interface module of MFX Engine.

Its logic functions include:

- MMIO register programming interface
- DMA action for fetching of execlists and ring data from memory
- Management of the Head pointer for the Ring Buffer
- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine
- Handling of user interrupts
- Handling of ring context switch interrupt
- Flushing the MFX Engine
- Handle NOP

The register programming (RM) bus is a DWord interface bus that is driven by the Gx Command Streamer. The VCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x4000 to 0x4FFFF. The Gx and MFX Engines use semaphore to synchronize their operations.

VCS operates completely independent of the Gx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside VCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (16 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards AVC/VC1/MPEG2 engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

PR_CTR_CTL - Watchdog Counter Control

PR_CTR_THRSH - Watchdog Counter Threshold

Logical Context Support

This section contains the registers for Logical Context Support.

Register
BB_STATE - Batch Buffer State Register
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
BB_ADDR - Batch Buffer Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
FORCE_TO_NONPRIV - FORCE_TO_NONPRIV
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2

Mode Registers

The following are Mode Registers:

Mode Register
MI_MODE - Mode Register for Software Interface
INSTPM - Instruction Parser Mode Register
NOPID - NOP Identification Register
IDLEDLY - Idle Switch Delay
RESET_CTRL - Reset Control Register
PREEMPTION_HINT - Preemption Hint
PREEMPTION_HINT_UDW - Preemption Hint Upper DWord
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait

Misc Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

Registers in Media Engine

This topic describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. The functions performed by some of these registers are discussed in more detail in the Memory Interface Functions, Memory Interface Instructions, and Programming Environment chapters.

The registers detailed in this chapter are used across multiple projects and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

GFX Pending TLB Cycles Information Registers

The following registers contain information about cycles that did not complete their TLB translation.

Information is organized as 64 entries, where each entry has a valid and ready bit, collapsed into separate registers.

TIMESTAMP - Reported Timestamp Count

CTX_TIMESTAMP - Context Timestamp Count

Memory Interface Commands for Video Codec Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

MI Commands
MI_ARB_CHECK
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_BATCH_BUFFER_START
MI_FLUSH_DW
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_MATH
MI_NOOP
MI_REPORT_HEAD
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_STORE_REGISTER_MEM
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_LOAD_REGISTER_MEM
MI_ATOMIC
MI_FORCE_WAKEUP

Video Enhancement Engine Command Interface

The following topics describe the Video Enhancement Engine Command Interface.

VECS_RINGBUF — Ring Buffer Registers

The following are Ring Buffer Registers:

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Logical Context Support

The following are Logical Context Support Registers:

Register
BB_ADDR - Batch Buffer Head Pointer Register
BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
BB_STATE - Batch Buffer State Register
TIMESTAMP - Reported Timestamp Count
CTX_TIMESTAMP - Context Timestamp Count
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
CCID - Current Context Register
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
FORCE_TO_NONPRIV - FORCE_TO_NONPRIV
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

Register
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2

Mode Registers

Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

MI Commands for Video Enhancement Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

MI Command
MI_ARB_CHECK
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_BATCH_BUFFER_START
MI_FLUSH_DW
MI_LOAD_REGISTER_REG
MI_MATH
MI_NOOP
MI_REPORT_HEAD
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_ATOMIC
MI_STORE_REGISTER_MEM
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_MEM

MI Command
MI_FORCE_WAKEUP

Watchdog Timers

Watchdog Counter Control

The Watchdog Counter Control determines if the watchdog is enabled, disabled and count mode. The watchdog is enabled is when the value of the register [30:0] is equal to zero([30:0] = 'd0). If enabled, then the Watchdog Counter is allowed to increment. The watchdog is disabled is when the value of the register [30:0] is equal to one where only bit zero is a value of '1'([30:0] = 0x00000001). If disabled, then the value of Watchdog Counter is reset to a value of zero. Bit 31, specifies the counting mode. If bit 31 is zero, then we will count based timestamp toggle(refer to Reported Timestamp Count register for toggle time). If bit 31 is one, then we will count every ungated GPU clock.

Programming Notes: When watch dog timer is enabled, HW does not trigger any kind of idle sequences. SW must enable and disable watch dog timer for any given workload within the same command buffer dispatch. SW must disable watch dog timer around semaphore waits and wait for events commands so that HW can trigger appropriate idle sequence for power savings.

This register is context saved as part of engine context.

Watchdog Counter Threshold

If the Watchdog Counter Threshhold is equal to Watchdog Counter, then the interrupt bit is set in the IIR(bit 6) and the Watchdog Counter is reset to zero.

This register is context saved as part of engine context.

Watchdog Counter

The Watchdog Counter is the count value of the watchdog timer. The Counter can be reset due to the Watchdog Counter Control being disabled or being equal to the Watchdog Counter Threshhold. The increment of the Watchdog counter is enabled when the Watchdog Counter Control is enabled and the current context is valid and execlist is enabled which includes the time to execute, flush and save the context.

The increment of the Watchdog counter is under the following conditions:

- Watchdog timer is enabled.
- Context is valid

The increment granularity is based controlled by Watchdog Counter Control mode(bit 31).

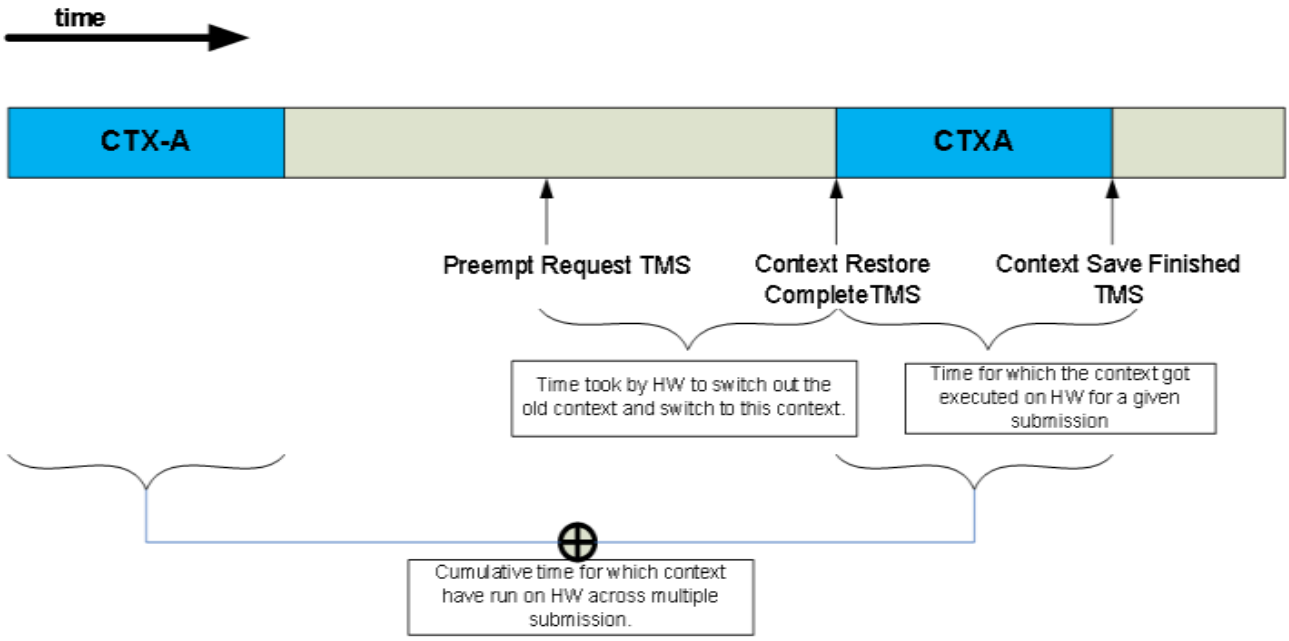
This register is not context saved and restored.

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

Figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Preemption

Preemption is a means by which HW is instructed to stop executing an ongoing workload and switch to the new workload submitted. Preemption flows are different based on the mode of scheduling.

Ring Buffer Scheduling

In Ring Buffer mode of scheduling SW triggers preemption by programming UHPTR (Updated Head Pointer Register) register with a valid head pointer. UHPTR contains head pointer and head pointer valid bit; the head pointer is valid only when the head pointer valid bit is set.

HW triggers preemption on a preemptable command on detecting Head Pointer Valid bit asserted in the UHPTR register. Following preemption HW updates its current head pointer with the Head Pointer from the UHPTR and starts execution; i.e all the commands from current head pointer to the updated head pointer are skipped by HW. HW samples the head pointer and the batch buffer address on preemption and updates them to the RING_BUFFER_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR respectively. RING_BUFFER_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR provide the graphics memory address of the preemptable command on which last preemption has occurred. HW resets the head pointer valid bit in UHPTR upon completion of preemption.

Programming Notes:

Preemption is not supported for Media Workloads. Hence preemption can be achieved only on Command Buffer boundaries. Media Command Buffers must be bracketed with MI_ARB_OFF and MI_ARB_ON commands to avoid preemption of media command buffers.

Example:

```
Ring Buffer
.
.
.
MI_ARB_ON_OFF     // OFF
MI_BATCH_START    // Media Workload
MI_ARB_ON_OFF     // ON
MI_ARB_CHK        // Preemptable command outside media command buffer.
.
.
End Ring Buffer
```

The following tables list the Preemptable Commands in the Ring Buffer mode of scheduling:

Engine (below)	Preemptable Commands				
	MI_ARB_CHECK	3DPRIMITIVE	GPGPU_WALKER	PIPE_CONTROL ***	MEDIA STATE FLUSH
Render	AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	N/A	N/A	N/A	N/A
Media	AP	N/A	N/A	N/A	N/A
VideoEnhancement	AP	N/A	N/A	N/A	N/A

AP: Allow preemption on UHPTR valid and arbitration enabled. Arbitration can be enabled/disabled using MI_ARB_ON_OFF command.

ExecList Scheduling

In ExecList mode of scheduling SW triggers preemption by submitting a new pending execlist to ELSP (ExecList Submit Port). HW triggers preemption on a preemptable command on detecting the availability of the new pending execlist, following preemption context switch happens to the newly submitted execlist. As part of the context switch preempted context state is saved to the preempted context LRCA, context state contains the details such that on resubmission of the preempted context HW can resume execution from the point where it was preempted.

Example:

```
Ring Buffer

MI_ARB_ON_OFF    // OFF
MI_BATCH_START   // Media Workload
MI_ARB_ON_OFF    // ON
MI_ARB_CHK       // Preemptable command outside media command buffer.
```

The following tables list the Preemptable Commands in ExecList mode of scheduling:

Engine (below)	Preemptable Commands							
	MI_ARB_CHECK	Element Boundary	Semaphore Wait	Wait for Event	3DPRIMITIVE	GPGPU_WALKER	PIPE_CONTROL ***	MEDIA STATE FLUSH
Render	AP	AP	Unsuccessful & AP	Unsuccessful & AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	AP	Unsuccessful & AP	Unsuccessful & AP	N/A	N/A	N/A	N/A
Media	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A
Video Enhancement	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A

Preemption is not supported for Media Workloads. Hence preemption can be achieved only on Command Buffer boundaries. Media Command Buffers must be bracketed with MI_ARB_OFF and MI_ARB_ON command to avoid preemption of media command buffers.

Table Notes:

AP - Allow Preemption if arbitration is enabled.

* 0x229c bit 11 determines whether the level of preemption is command or object level.

** 0x20E4 bits 2:1 determine the level of preemption for GPGPU workloads.

*** MI_ATOMIC and MI_SEMAPHORE_SIGNAL commands with Post Sync Op bit set are treated as PIPE_CONTROL command with Post Sync Operation as Atomics or Semaphore Signal.

Command Streamer (CS) ALU Programming

The command streamer implements a rudimentary Arithmetic Logic Unit (ALU) which supports basic arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64-bit operands.

The ALU has two 64-bit registers at the input, SRCA and SRCB, to which source operands are loaded. The ALU result is written to a 64-bit accumulator. The Zero Flag and Carry Flag are assigned based on the accumulator output.

See the ALU Programming section in the Render Engine Command Streamer, for a description of the ALU programming model. Programming model is the same for all command streamers that support ALU, but

each command streamer uses its own MMIO address range to address the registers. The following subsections describe the ALU registers and the programming details.

CS ALU Programming and Design

MI Commands for Graphics Processing Engines

This chapter lists the MI Commands that are supported by Generic Command Streamer Front End implemented in the graphics processing engines (Render, Video, Blitter and Video Enhancement).

Command
MI_NOOP
MI_ARB_CHECK
MI_BATCH_BUFFER_START
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_DISPLAY_FLIP
MI_LOAD_SCAN_LINES_EXCL
MI_LOAD_SCAN_LINES_INCL
MI_CLFLUSH
MI_MATH
MI_REPORT_HEAD
MI_STORE_DATA_IMM
MI_STORE_DATA_INDEX
MI_ATOMIC
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_LOAD_REGISTER_MEM
MI_STORE_REGISTER_MEM
MI_USER_INTERRUPT
MI_WAIT_FOR_EVENT
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as “Command Parser Master Error” interrupt to SW. Privilege access violation checks in HW can be disabled

by setting “Privilege Check Disable” bit in GFX_MODE register. When privilege access checks are disabled HW executes the Privilege command as expected.

User Mode Privileged Command	Function in Non-Privileged Batch Buffers	Source
MI_UPDATE_GTT	Command is converted to NOOP.	*CS
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_STORE_DATA_INDEX	Command is converted to NOOP.	*CS
MI_STORE_REGISTER_MEM	Register read is always performed. Memory update is dropped if **Use Global GTT** is enabled.	*CS
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its “Privileged” level to its parent batch buffer or lower. Chained or Second level batch buffer can be “Privileged” only if the parent or the initial batch buffer is “Privileged”. This is HW enforced.	*CS
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.	*CS
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Command is converted to NOOP if the register accessed is privileged.	*CS
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** is discarded.	*CS
MI_REPORT_PERF_COUNT	Command is converted to NOOP if **Use Global GTT** is enabled.	Render CS
PIPE_CONTROL	Still send flush down, Post-Sync Operation is NOOP if **Use Global GTT** or Use “Store Data Index” is enabled. Post-Sync Operation LRI to **Privileged Register** is discarded.	Render CS
MI_SET_CONTEXT	Command is converted to NOOP.	Render CS
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.	Render CS
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.	*CS
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_ARB_ON_OFF	Command is converted to NOOP.	*CS
MI_DISPLAY_FLIP	Command is converted to NOOP.	*CS

User Mode Privileged Command	Function in Non-Privileged Batch Buffers	Source
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.	*CS
MI_FLUSH_DW	Still send flush down, Post-Sync Operation is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.	Blitter CS, Video CS, Video Enhancement CS

Parsing one of the commands in the table above from a non-privileged batch buffer flags an error and converts the command to a NOOP.

The tables below list the non-privileged registers that can be written to from a non-privileged batch buffer executed from various command streamers.

User Mode Non-Privileged Registers for Render Command Streamer (RCS)

MMIO Name	MMIO Offset	Size in DWords
Cache_Mode_0	0x7000	1
Cache_Mode_1	0x7004	1
GT_MODE	0x7008	1
L3_Config	0x7034	1
TD_CTL	0xE400	1
TD_CTL2	0xE404	1
L3SQCREG4	0xB118	1
NOPID	0x2094	1
INSTPM	0x20C0	1
IA_VERTICES_COUNT	0x2310	2
IA_PRIMITIVES_COUNT	0x2318	2
VS_INVOCATION_COUNT	0x2320	2
HS_INVOCATION_COUNT	0x2300	2
DS_INVOCATION_COUNT	0x2308	2
GS_INVOCATION_COUNT	0x2328	2
GS_PRIMITIVES_COUNT	0x2330	2
SO_NUM_PRIMS_WRITTEN0	0x5200	2
SO_NUM_PRIMS_WRITTEN1	0x5208	2
SO_NUM_PRIMS_WRITTEN2	0x5210	2
SO_NUM_PRIMS_WRITTEN3	0x5218	2
SO_PRIM_STORAGE_NEEDED0	0x5240	2
SO_PRIM_STORAGE_NEEDED1	0x5248	2
SO_PRIM_STORAGE_NEEDED2	0x5250	2
SO_PRIM_STORAGE_NEEDED3	0x5258	2

MMIO Name	MMIO Offset	Size in DWords
SO_WRITE_OFFSET0	0x5280	1
SO_WRITE_OFFSET1	0x5284	1
SO_WRITE_OFFSET2	0x5288	1
SO_WRITE_OFFSET3	0x528C	1
CL_INVOCATION_COUNT	0x2338	2
CL_PRIMITIVES_COUNT	0x2340	2
PS_INVOCATION_COUNT_0	0x22C8	2
PS_DEPTH_COUNT _0	0x22D8	2
PS_INVOCATION_COUNT_1	0x22F0	2
PS_DEPTH_COUNT _1	0x22F8	2
PS_INVOCATION_COUNT_2	0x2448	2
PS_DEPTH_COUNT_2	0x2450	2
GPUGPU_DISPATCHDIMX	0x2500	1
GPUGPU_DISPATCHDIMY	0x2504	1
GPUGPU_DISPATCHDIMZ	0x2508	1
MI_PREDICATE_SRC0	0x2400	1
MI_PREDICATE_SRC0	0x2404	1
MI_PREDICATE_SRC1	0x2408	1
MI_PREDICATE_SRC1	0x240C	1
MI_PREDICATE_DATA	0x2410	1
MI_PREDICATE_DATA	0x2414	1
MI_PREDICATE_RESULT	0x2418	1
MI_PREDICATE_RESULT_1	0x241C	1
MI_PREDICATE_RESULT_2	0x23BC	1
3DPRIM_END_OFFSET	0x2420	1
3DPRIM_START_VERTEX	0x2430	1
3DPRIM_VERTEX_COUNT	0x2434	1
3DPRIM_INSTANCE_COUNT	0x2438	1
3DPRIM_START_INSTANCE	0x243C	1
3DPRIM_BASE_VERTEX	0x2440	1
GPGPU_THREADS_DISPATCHED	0x2290	2
BB_OFFSET	0x2158	1
CS_GPR (1-16)	0x2600	32
OA_CTX_CONTROL	0x2360	1
OACTXID	0x2364	1
OA CONTROL	0x2B00	1

MMIO Name	MMIO Offset	Size in DWords
PERF_CNT_1_DW0	0x91b8	1
PERF_CNT_1_DW1	0x91bc	1
PERF_CNT_2_DW0	0x91c0	1
PERF_CNT_2_DW1	0x91c4	1

User Mode Non-Privileged Registers for Blitter Command Streamer (BCS)

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	0x22600	32
BCS_SWCTRL	0x22200	1

This table represents the Base offset for Video Command Streamers and Media Engine message range:

Unit	MMIO Base Offset	Description
VCS/MFC	0x12000	Video Command Streamer 0
VCS1/MFC1	0x1C000	Video Command Streamer 1
VECS	0x1A000	Video Enhancement Command Streamer
HEVC	0x1E900	

User Mode Non-Privileged Registers for Video Enhancement Command Streamer (VECS)

MMIO Name	MMIO Base	MMIO Offset	Size in DWords
VECS_GPR	VECS	0x600	32

User Mode Non-Privileged Registers for Video Command Streamer (ALL VCS)

MMIO Name	Unit Base	MMIO Range	Size in DWords
VCS_GPR	VCS	0x600	32
MFC_VDBOX1	VCS	0x800-0xFFF	512
HEVC	HEVC	0x00	64
HEVC-Enc	HEVC	0x00	64
Compression IP Range	Compression IP	0x00	256

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 7: 3D-Media-GPGPU

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

3D-Compute Front End and Software Interface 1

Predication 1

Predicate Render Registers 1

MI_SET_PREDICATE 1

MI_PREDICATE 2

Predicated Rendering Support in HW 2

3D-Compute Front End and Software Interface 4

VF Instance Count Registers 4

Mode and Misc Ctrl Registers 5

Pipelines Statistics Counter Registers 6

AUTO_DRAW Registers 7

MMIO Registers for GPGPU Indirect Dispatch 7

CS ALU Programming and Design 8

CS ALU Programming and Design 8

Generic Purpose Registers 8

ALU BLOCK Diagram 9

Instruction Set 9

Instruction Format 10

LOAD Operation 10

Arithmetic/Logical Operations 10

STORE Operation 11

Summary for ALU 11

Summary of Instructions Supported 12

Table for ALU OPCODE Encodings 12

Table for Register Encodings 13

CS_GPR - Command Streamer General Purpose Registers 14

Memory Interface Commands for Rendering Engine 14

State Commands 15

STATE_BASE_ADDRESS 15

Synchronization of the 3D Pipeline 15

Top-of-Pipe Synchronization 16

End-of-Pipe Synchronization 16
Synchronization Actions 17
Writing a Value to Memory 17
PS_DEPTH_COUNT 17
Generating an Interrupt 18
Invalidating of Caches 18
PIPE_CONTROL Command 18
PIPE_CONTROL 19
Programming Restrictions for PIPE_CONTROL 19
Post-Sync Operation 20
Flush Types 20
Stall 21
Context Image 21
Power Context Image 21
Render Engine Power Context 21
Engine Register and State Context 23
Register State Context 24
Command Ordering Rules 41
PIPELINE_SELECT 41
PIPE_CONTROL 41
Common Pipeline State-Setting Commands 41
3D Pipeline-Specific State-Setting Commands 42
Media Pipeline-Specific State-Setting Commands 42
3DPRIMITIVE 42
MEDIA_OBJECT 42
Resource Streamer 43
Resource Streamer Sync Commands 43
Introduction 43
Glossary 43
Common Abbreviations 44
Theory of Operation 44
Resource Streamer Functions 44
Detailed Resource Streamer Operations 46

Introduction 46
Resource Streamer Operation Descriptions 46
Batch Processing 46
Context Save 46
HW Binding Table Image 47
Gather Push Constants Image 47
Push Constant Image 48
HW Binding Table Generation 49
Gather Push Constants 49
Constant Buffer Generation (not DX9) 50
Commands Actions in the RS 51
Resource Streamer Programming Guidelines 58
RS Interactions with the 3D Command Streamer 58
RS Interactions with Memory Requests 58
Fundamental Programming and Operational Assumptions 58
Non-Operational Activities 59
Hardware Binding Tables 59
3DSTATE_BINDING_TABLE_POOL_ALLOC 59
Gather Constants 60
Dx9 Constant Buffer Generation 61
Vertex Shader Constant 61
Pixel Shader Constant 62
Shared Functions 62
Binding Table 63
3D Sampler 65
Sampling Engine 66
3D Sampler Theory of Operation 66
Sampler Inputs Messages 67
Sampler Data Fetches 68
Sampler Filtering and Processing 68
Texture Coordinate Processing 69
Texture Coordinate Normalization 69
Texture Coordinate Computation 69

Sampler SW performance hints 71
Texel Address Generation 72
Level of Detail Computation (Mipmapping) 72
Base Level Of Detail (LOD) 73
LOD Bias 73
LOD Pre-Clamping 74
Min/Mag Determination 74
LOD Computation Pseudocode 75
Intra-Level Filtering Setup 76
MAPFILTER_NEAREST 77
MAPFILTER_LINEAR 77
Bilinear Filter Sampling 78
MAPFILTER_ANISOTROPIC 78
MAPFILTER_MONO 79
Inter-Level Filtering Setup 80
Texture Address Control 81
TEXCOORDMODE_MIRROR Mode 82
TEXCOORDMODE_MIRROR_ONCE Mode 83
TEXCOORDMODE_WRAP Mode 84
TEXCOORDMODE_CLAMP Mode 85
TEXCOORDMODE_CLAMPBORDER Mode 86
TEXCOORDMODE_HALF_BORDER Mode 86
TEXCOORDMODE_CUBE Mode 86
Texel Fetch 87
Texel Chroma Keying 87
Chroma Key Testing 87
Chroma Key Effects 87
Shadow Prefilter Compare 88
Texel Filtering 88
Texel Color Gamma Linearization 89
Multisampled Surface Behavior 89
Multisample Control Surface 89
Quilted Textures] 90

State 91
Surface State Fetch 91
Bindless Surface State Fetch 91
Sampler State Fetch 91
Bindless Sampler State 92
State Caching 92
SURFACE_STATE 93
Surface Formats 93
Sampler Output Channel Mapping 96
SURFACE_STATE for Deinterlace sample_8x8 and VME 102
SAMPLER_STATE 103
Border Color Programming for Integer Surface Formats 105
Messages 106
Message Descriptor and Execution Mask 107
Execution Mask 107
Message Descriptor 107
Restrictions for Message Descriptors 108
Message Header 108
Message Types 113
3D Sampler Message Types 113
Common Message Variants 113
Restrictions and Programming Notes for All Message Types 114
Sample Message Types 114
ld Message Types 118
gather4 Message Types 121
Definitions 121
Supported Variants: 121
Restrictions and Programming Notes for gather4: 122
Restrictions and Programming Notes for gather4_c: 122
Restrictions and Programming Notes for gather4_po: 123
Restrictions and Programming Notes for gather4_po_c: 123
sampleinfo Message Type 124
sampleinfo Message Definition 124

Supported Variants:.....124
Restrictions and Programming Notes for sampleinfo:.....124
LOD Message Type.....124
LOD Message Definition.....124
Supported Variants:.....124
Restrictions and Programming Notes for LOD:.....125
resinfo Message Type.....125
resinfo Message Definition.....125
Supported Variants:.....126
Restrictions and Programming Notes for resinfo:.....126
cache_flush Message Type.....126
cache_flush Message Definition.....126
Supported Variants:.....126
Media Message Types.....127
sample_unorm Message Types.....127
sample_unorm Message Definition.....127
Supported Variants:.....127
Restrictions and Programming Notes for sample_unorm, sample_unorm_RG, sample_unorm_killpix, sample_unorm_RG_killpix:.....128
sample_8x8 Message Type.....129
Supported Variants:.....129
Restrictions and Programming Notes for sample_8x8:.....129
Message Format.....129
Supported SIMD Types.....129
Message Length.....130
Response Length.....130
Message Formats.....133
Parameter Types.....133
SIMD Payloads.....135
Writeback Message.....138
SIMD16.....138
Return Format = 32-bit.....138
Return Format = 16-bit.....140

SIMD8 141
Return Format = 32-bit 141
Return Format = 16-bit 142
SIMD4x2 143
Direct Write To Render Target 144
Shared Functions – Data Port 145
Read/Write Ordering 146
Address Models 147
Binding Table Surface State Model (SSM and BTS) 148
A32 Stateless Model 148
A64 Stateless Model 150
Shared Local Memory (SLM) 151
Address Models during SIP 153
Overview of Memory Accesses 154
Surfaces Types 156
Addressing Buffers (SURFTYPE_BUFFER) 157
Addressing Structured Buffers (SURFTYPE_STRBUF) 157
Addressing 1D, 2D, 3D, CUBE Surfaces 158
Surface Formats 161
Addressing 2D Media Surfaces 163
2D Media Surface Formats 166
Slots and Elements 168
Execution Masks 170
Address Alignment and Data Widths 172
Bounds Checking and Faulting 173
Message Formats 175
End of Thread Usage 176
Message Description Conventions 176
Messages 177
Scattered and Block Read/Write Messages 181
Byte Scattered ReadWrite Messages 181
Byte Scaled Read/Write Messages 184
DWord Scattered Read/Write Messages 185

DWord Scaled Read/Write Messages 187
QWord Scattered Read/Write Messages 188
QWord Scaled Read/Write Messages 189
OWord Block Read/Write Messages 190
OWord Aligned Block Read/Write Messages 191
OWord Dual Block Read/Write Messages 193
HWord Scratch Block Read/Write Messages 194
HWord Aligned Block Read/Write Messages 195
Surface ReadWrite Messages 196
Untyped Surface ReadWrite Messages 196
Scaled Untyped Surface Read/Write Messages 200
Typed Surface Read/Write Messages 203
MSAA Typed Surface ReadWrite Messages 204
Atomic Operation Messages 205
DWord Untyped Atomic Integer Messages 206
DWord Scaled Untyped Atomic Integer Messages 211
QWord Untyped Atomic Integer Messages 213
QWord Scaled Untyped Atomic Integer Messages 216
OWord Untyped Atomic Integer Messages 217
DWord Untyped Atomic Float Messages 217
DWord Scaled Untyped Atomic Float Messages 221
DWord Typed Atomic Integer Messages 222
DWord Atomic Counter Messages 225
Media Surface Read/Write Messages 226
Transpose Read Message 226
Media Block Read/Write Messages 228
Pixel Masked Media Block Write Message 230
Byte Masked Media Block Write Message 231
Merged Media Block Read/Write Messages 233
Other Data Port Messages 238
Read Surface Info Message 238
Memory Fence Message 239
Message-Specific Descriptors 240

Data Port 0 Message Specific Descriptors 240
Scattered Read/Write Messages 241
Block Read/Write Messages 241
Other Data Port Messages 241
Data Port 1 Message Specific Descriptors 241
A64 Scattered Read/Write 241
A64 Block Read/Write 242
Surface Read/Write 242
Media Messages 242
A32 Dword Untyped Atomic Operations 242
A64 Dword Untyped Atomic Operations 243
A64 Qword Untyped Atomic Operations 244
Dword Typed Atomic Operations 244
Dword Atomic Counter Operations 244
Data Port 2 Message Specific Descriptors 245
A32 and SLM Scaled Read/Write 245
A64 Scaled Read/Write 245
Read-Only Data Port Message Specific Descriptors 245
Block Read/Write 245
Other Messages 246
Message Headers 246
Data Port 0 Message Headers 247
Data Port 1 Message Headers 247
Data Port 2 Message Headers 247
Message Address Payloads 248
32 Bit Address Payloads 248
64 Bit Address Payloads 248
32 Bit Untyped Surface Address Payloads 248
64 Bit Untyped Surface Address Payloads 248
32 Bit Typed Surface Address Payloads 248
Message Data Payloads 249
Oword Block Data Payloads 249
Hword Block Data Payloads 249

Dword SIMD Data Payloads 249
Qword SIMD Data Payloads 249
SIMD Atomic Operation Data Payloads 250
Other Data Payloads 250
Common Message Descriptor Controls 250
Binding Table Index Message Descriptor Control Fields 250
Header Present Message Descriptor Control Fields 251
Data Blocks Message Descriptor Control Fields 251
SIMD Mode Message Descriptor Control Fields 251
Atomic Operation Message Descriptor Control Fields 251
Other Message Descriptor Control Fields 252
Common Message Payload Controls 252
Header Controls 252
Address Controls 252
Data Controls 253
Pixel Data Port 253
Cache Agents 253
Shared Functions - Render Target 253
Render Target 254
Render Target Surfaces 254
Render Target Surface Types 254
Surface Formats for Render Target Messages 255
Subspan/Pixel to Slot Mapping 256
Message Common Control Fields 259
Render Target Messages 260
Render Target Write Message 260
Replicate Data 266
Single Source 266
Dual Source 267
Render Target Read Message 267
Message-Specific Descriptors 268
Message Headers 268
Message Data Payloads 268

Render Target Data Payloads 269
Render Target 271
Multiple Render Targets (MRT) 271
Flushing the Render Cache 271
End of Thread Usage 271
Execution Mask 271
Bounds Checking 272
Shared Functions Pixel Interpolater 272
Messages 273
Initiating Message 273
Message Descriptor 273
"Per Message Offset" Message Descriptor 275
"Sample Position Offset" Message Descriptor 275
"Centroid Position" and "Per Slot Offset" Message Descriptor 275
Message Payload for Most Messages 276
SIMD8 Per Slot Offset Message Payload 276
SIMD16 Per Slot Offset Message Payload 277
Writeback Message 279
SIMD8 279
SIMD16 280
Shared Functions - Unified Return Buffer (URB) 282
URB Size 282
URB Access 282
State 283
URB Messages 283
Execution Mask 284
Message Descriptor 284
URB_WRITE and URB_READ 286
Message Header 286
URB_WRITE_HWORD Write Data Payload 287
URB_NOSWIZZLE 288
URB_INTERLEAVED 289
URB_READ_HWORD Writeback Message 291

URB_NOSWIZZLE 291
URB_INTERLEAVED 293
URB_WRITE_OWORD Write Data Payload 294
URB_NOSWIZZLE 294
URB_INTERLEAVED 296
URB_READ_OWORD Writeback Message 296
URB_NOSWIZZLE 297
URB_INTERLEAVED 298
URB_ATOMIC 298
Message Header 299
Writeback Message 299
URB_SIMD8_Write and URB_SIMD8_Read 300
Message Descriptor 300
Message Header 301
Per Slot Offset Message Phase 301
Channel Mask Message Phase 303
Write Data Payload 305
Writeback Message 306
Message Gateway 307
Message Descriptor 307
GetTimeStamp Message 307
Message Payload 308
Writeback Message to Requester Thread 308
BarrierMsg Message 309
Message Payload 310
Writeback Message to Requester Thread 311
Broadcast Writeback Message 311
Media Sampler 312
Shared Functions – Video Motion Estimation 312
Theory of Operation 312
Shape Decision 312
Major Shape Decision Prior to FME 313
Shape Update after FME 314

Final Code Decision after BME 314
Surfaces 314
State 314
BINDING_TABLE_STATE 314
SURFACE_STATE 314
VME_STATE 315
VME_SEARCH_PATH_LUT_STATE 315
Glossary of Messages 320
Universal Input Message Phases 320
SIC Input Message Phases 344
IME Input Message Phases 350
FBR Input Message Phases 355
IDM Input Message Phases 357
Return Data Message Phases 358
IME StreamOut 372
IDM Stream-Out 372
IDM16x16 Streamout Message Format 373
IDM8x8 Streamout Message Format 374
Sample_8x8 State 376
SURFACE_STATE for Deinterlace, sample_8x8, and VME 376
SAMPLER_STATE for Sample_8x8 Message 376
SIMD32/64 Messages 378
Initiating Message 378
SIMD32/64 Payload 378
SIMD32 Payload 378
SIMD64 Payload 379
Vertical Block Number Restrictions 383
Payload Parameter Definition 384
SIMD32_64 Message Descriptor 387
SIMD32_64 Message Header 387
Message Header 387
SIMD32_64 Payload Parameter Definition 391
SIMD32_64 Message Types 391

Writeback Message 391
SIMD32 391
Sample_unorm* 392
Cache_flush 399
Sample_8x8 Writeback Messages 399
"16 bit Full" Output Format Control Mode 400
Sampler_8x8 – Writeback Message for AVS 407
"16 bit Full" Output Format Control Mode 408
"16 Bit Chrominance Downsampled" Output Format Control Mode 409
"8 Bit Full" Output Format Control Mode 414
"8 Bit Chrominance Downsampled" Output Format Control Mode 416
SIMD32 Surface State 420
SIMD32 Sampler State 420
3D Pipeline Stages 420
3D Pipeline-Level State 421
3D Pipeline Geometry 423
Block Diagram 423
3D Primitives Overview 425
Thread Request Generation 432
Thread Control Information 432
Thread Payload Generation 433
Fixed Payload Header 434
Extended Payload Header 436
Payload URB Data 436
Vertex Data Overview 438
Vertex URB Entry (VUE) Formats 438
Vertex Positions 441
Clip Space Position 442
NDC Space Position 442
Screen-Space Position 442
Vertex Fetch (VF) Stage 443
State 443
Control State 443

Index Buffer (IB) State 443
Vertex Buffers (VB) State 444
VERTEXDATA Buffers – SEQUENTIAL Access 444
VERTEXDATA Buffers – RANDOM Access 445
INSTANCEDATA Buffers 446
Vertex Definition State 447
Input Vertex Definition 447
3D Primitive Command 448
Functions 448
Input Assembly 448
Vertex Assembly 448
Vertex Cache 449
Input Data: Push Model vs. Pull Model 450
Generated IDs 450
3D Primitive Processing 451
Index Buffer Access 451
Vertex Element Data Path 452
FormatConversion 453
DestinationFormatSelection 457
Dangling Vertex Removal 457
Vertices Generated 457
Objects Generated 458
Vertex Shader (VS) Stage 458
State 458
Payloads 458
SIMD4x2 Payload 458
SIMD8 Payload 460
Hull Shader (HS) Stage 463
State 463
Functions 464
Patch Object Staging 464
HS Thread Execution 464
HS Thread Dispatch Mask 464

Patch URB Entry (Patch Record) Output 465
Patch Header DW0-7 465
Statistics Gathering 466
HS Invocations 466
Payloads 467
SINGLE_PATCH Payload 467
DUAL_PATCH Payload 471
8_PATCH Payload 473
Tessellation Engine (TE) Stage 477
State 478
Functions 478
Patch Culling 478
Tessellation Factor Limits 478
Partitioning 478
Domain Types and Output Topologies 479
QUAD Domain Tessellation 479
TRI Domain Tessellation 480
ISOLINE Domain Tessellation 481
Domain Shader (DS) Stage 482
State 482
Functions 483
SIMD4x2 Thread Execution 483
DUAL_PATCH Thread Execution 483
Statistics Gathering 484
Payloads 484
SIMD4x2 Payload 484
SIMD8 Payload 487
DUAL_PATCH Payload 490
Geometry Shader (GS) Stage 493
GS Stage Overview 493
State 493
Functions 494
Object Staging 494

Thread Request Generation 494
Object Vertex Ordering 494
Thread Execution 499
Thread Execution 500
GS URB Entry 500
GS URB Entry - Output Vertex Count 501
GS Output Topologies 502
GS Output StreamID 503
Primitive Output 503
Statistics Gathering 503
Payloads 504
Thread Payload High-Level Layout 504
SIMD 4x2 Thread Payload 505
SIMD8 Thread Payload 514
Stream Output Logic (SOL) Stage 519
State 519
Functions 521
Input Buffering 521
Stream Output Function 524
Stream Output Buffers 525
Rendering Disable 526
Statistics 526
3D Pipeline Rasterization 526
Common Rasterization State 526
3D Pipeline – CLIP Stage Overview 526
Clip Stage – 3D Clipping 527
Fixed Function Clipper 527
Concepts 527
CLIP Stage Input 527
State 528
VUE Readback 529
VertexClipTest Function 530
Object Staging 533

Partial Object Removal 533
ClipDetermination Function 533
ClipMode State 535
NORMAL ClipMode 536
CLIP_ALL ClipMode 536
CLIP_NON_REJECT ClipMode 536
REJECT_ALL ClipMode 536
ACCEPT_ALL ClipMode 537
Object Pass-Through 537
Primitive Output 538
Other Functionality 538
Statistics Gathering 538
CL_INVOCATION_COUNT 539
3D Pipeline - Strips and Fans (SF) Stage 539
Inputs from CLIP 539
Attribute Setup/Interpolation Process 540
Attribute Setup/Interpolation Process 540
Outputs to WM 541
Primitive Assembly 541
Point List Decomposition 544
Line List Decomposition 545
Line Strip Decomposition 546
Triangle List Decomposition 548
Triangle Strip Decomposition 549
Triangle Fan Decomposition 550
Polygon Decomposition 551
Rectangle List Decomposition 551
Object Setup 553
Invalid Position Culling (Pre/Post-Transform) 553
Viewport Transformation 553
Destination Origin Bias 553
Point Rasterization Rule Adjustment 554
Drawing Rectangle Offset Application 555

Point Width Application 557
Rectangle Completion 558
Vertex XY Clamping and Quantization 558
Degenerate Object Culling 559
Triangle Orientation (Face) Culling 559
Scissor Rectangle Clipping 560
Viewport Extents Test 561
Line Rasterization 562
Zero-Width (Cosmetic) Line Rasterization 562
GIQ (Diamond) Sampling Rules – Legacy Mode 562
GIQ (Diamond) Sampling Rules – DX10 Mode 564
Non-Antialiased Wide Line Rasterization 566
Anti-Aliased Line Rasterization 567
SF Pipeline State Summary 568
3DSTATE_RASTER 568
3DSTATE_SF 569
Attribute Interpolation Setup 574
Attribute Swizzling 574
Interpolation Modes 575
Point Sprites 575
Barycentric Attribute Interpolation 577
Depth Offset 577
Other SF Functions 577
Statistics Gathering 577
Windower (WM) Stage 578
Overview 578
Inputs from SF to WM 578
Windower Pipelined State 579
3DSTATE_WM 579
3DSTATE_SAMPLE_MASK 584
Rasterization 590
Drawing Rectangle Clipping 591
Line Rasterization 591

Coverage Values for Anti-Aliased Lines 592
3DSTATE_AA_LINE_PARAMS 592
Line Stipple 592
Polygon (Triangle and Rectangle) Rasterization 593
Polygon Stipple 594
Multisampling 594
Multisample ModesState 595
Other WM Functions 600
Statistics Gathering 600
Pixel 600
Early Depth/Stencil Processing 601
Depth Offset 601
Early Depth Test/Stencil Test/Write 602
Software-Provided PS Kernel Info 602
Hierarchical Depth Buffer 603
Depth Buffer Clear 604
Depth Buffer Resolve 605
Hierarchical Depth Buffer Resolve 606
Optimized Depth Buffer Resolve 606
Optimized Hierarchical Depth Buffer Resolve 606
Separate Stencil Buffer 607
DepthStencil Buffer State 607
Pixel Shader Thread Generation 608
3DSTATE_PS 608
Pixel Grouping (Dispatch Size) Control 610
Multisampling Effects on Pixel Shader Dispatch 612
MSDISPMODE_PERPIXEL Thread Dispatch 612
MSDISPMODE_PERSAMPLE Thread Dispatch 612
PS Thread Payload for Normal Dispatch 616
PS Thread Payload for Normal Dispatch 617
Pixel Backend 630
Color Calculator (Output Merger) 630
Overview 630

Alpha Coverage 631
Alpha Test 631
Depth Coordinate Offset 632
Stencil Test 633
Depth Test 633
Pre-Blend Color Clamping 634
Pre-Blend Color Clamping When Blending is Disabled 635
Pre-Blend Color Clamping When Blending is Enabled 635
Color Buffer Blending 635
Post-Blend Color Clamping 637
Dithering 638
Logic Ops 639
Buffer Update 640
Stencil Buffer Updates 640
Depth Buffer Updates 641
Color Gamma Correction 641
Color Buffer Updates 641
Pixel Pipeline State Summary 642
COLOR_CALC_STATE 642
3DSTATE_BLEND_STATE_POINTERS 642
3DSTATE_DEPTH_STENCIL_STATE_POINTERS 642
COLOR_CALC_STATE 642
DEPTH_STENCIL_STATE 642
BLEND_STATE 642
CC_VIEWPORT 642
Other Pixel Pipeline Functions 643
Statistics Gathering 643
MCS Buffer for Render Target(s) 643
Render Target Fast Clear 646
Render Target Resolve 646
Media GPGPU Pipeline 647
Media GPGPU Pipeline] 647
Programming the GPGPU Pipeline 647

GPGPU Thread Limits 648
GPGPU Commands 648
GPGPU Command Workarounds 649
GPGPU Indirect Thread Dispatch 649
GPGPU Context Switch 650
GPGPU Context Switch 651
GPGPU Context Switch Workarounds 653
Media GPGPU Payload Limitations 653
Synchronization of the Media/GPGPU Pipeline 654
Mode of Operations 654
GPGPU Mode 654
Automatic Thread Generation 654
Thread Payload 655
Execution Masks 656
URB Management 658
Indirect Payload Storage 658
MEDIA_OBJECT_GRPID 660
Starting Offset for a Thread Group ID 661
GPGPU Thread R0 Header 662
Thread Group Tracking 664
Shared Local Memory Allocation 664
Software Managed Shared Local Memory 664
Automatic Barrier Management 665
Dispatch Payload 665
Cross-Slice Barriers 665
Generic Media 666
Media and General Purpose Pipeline 668
Introduction 668
Terminology 668
Hardware Feature Map in Products 670
Media Pipeline Overview 671
Generic Mode 672
Programming Media Pipeline 673

Command Sequence 673
Command Sequence 673
Parameterized Media Walker 675
Walker Parameter Description 676
Basic Parameters for the Local Loop 677
MbAff-Like Special Case in Local Loop 678
Global Loop 679
Walker Algorithm Description 680
Barriers and Shared Local Memory 684
Flexible Dispatch of Local Loop 685
Masked Dispatch 686
Scoreboard Control 686
AVC-Style Dependency Example 687
Interface Descriptor Selection 689
VC1-Style Dependency Example 691
Multiple Slice Considerations 691
Interrupt Latency 692
Thread Spawner Unit 692
Root Threads and Child Threads 694
Root Threads 694
URB Handles 694
Root to Child Responsibilities 695
Multiple Simultaneous Roots 695
Synchronized Root Threads 696
Deadlock Prevention 696
Child Thread Life Cycle 697
Arbitration between Root and Child Threads 698
Persistent Root Thread 698
Media State Model 699
Media State and Primitive Commands 699
Media State and Primitive Command Workarounds 700
Media Payload 700
Media Messages 701

Thread Payload Messages 701
Generic Mode Root Thread 702
Root Thread from MEDIA_OBJECT_PRT 703
Root Thread from MEDIA_OBJECT_WALKER 704
MEDIA_OBJECT_GRPID and MEDIA_OBJECT_WALKER with Groups Payload 704
Thread Spawn Message 706
Message Descriptor 707
Message Payload 708
Media-GPGPU Thread EOT Message 709
Message Descriptor 709
Message Payload 709
L3 Cache and URB 710
L3 Cache and URB 710
LBCF Registers 710
LNCF Registers 711
LPFC Registers 712
Overview 712
L3 Bank Configuration 713
Bandwidth and Throughput Capability 713
L3 Blocks Overview 714
Size of L3 Bank and Allocations 714
L3 Cache Theory of Operation 715
L3 Cache Theory of Operation 716
Access Policy 717
Arbitration Policies 717
Tag Request Arbitration 717
Data Request Arbitration 718
L3 Ordering Restrictions 718
Basic Ordering Requirements 718
Allocation Policy 720
High Level Algorithm Descriptions 720
High Level Algorithm Descriptions 720
Memory Object Control State on Cacheability 721

Overview 721
Atomics 722
L3 Coherency 726
Thread Level Coherency 727
Thread Group Coherency 727
GPUIA Level Coherency 727
Coherency Usage Models 727
Fixed Func Producing (URB) 728
Fixed Func Producing (Push Constants) 728
EUs Producing via HDC 728
Invalidation and Flushes 729
Node Invalidation Architecture 729
Command Streamer Invalidation Flows 730
Non-IA Coherent Flows 730
Top of the Pipe Invalidations 730
Pipeline Flush 730
IA-Coherent Flows 730
EUThread Flows 731
Global Invalidation 731
Power Management Invalidation 731
L3 Allocation and Programming 732
Shared Local Memory 735
Save and Restore Requirements 737
L3 Cache Error Protection 738
L3 Cache and URB 739
LBCF Registers 739
LNCF Registers 739
LPFC Registers 739
EU Overview 740
CoIssue/Dual Issue: 741
Thread scheduling: 741
Primary Usage Models 741
AOS and SOA Data Structures 741

SIMD4 Mode of Operation 743
SIMD4x2 Mode of Operation 744
SIMD16 Mode of Operation 746
SIMD8 Mode of Operation 747
Messages 748
Message Payload Containing a Header 749
Writebacks 749
Message Delivery Ordering Rules 750
Execution Mask and Messages 750
End-Of-Thread (EOT) Message 750
Performance 751
Message Description Syntax 752
Message Errors 752
Registers and Register Regions 754
Register Files 754
GRF Registers 754
ARF Registers 755
ARF Registers Overview 755
Access Granularity 756
Null Register 757
Address Register 757
Accumulator Registers 761
Flag Register 765
Channel Enable Register 766
Message Control Registers 767
Stack Pointer Register 768
State Register 769
Control Register 772
Notification Registers 778
IP Register 780
TDR Registers 781
Performance Registers 784
Flow Control Registers 785

Immediate 786
Region Parameters 787
Region Addressing Modes 791
Direct Register Addressing 791
Register-Indirect Register Addressing with a 1x1 Index Region 793
Register-Indirect Register Addressing with a Vx1 Index Region 794
Register-Indirect Register Addressing with a VxH Index Region 795
Access Modes 797
Execution Data Type 798
Register Region Restrictions 799
Destination Operand Description 804
Destination Region Parameters 804
SIMD Execution Control 805
Predication 805
No Predication 807
Predication with Horizontal Combination 807
Predication with Vertical Combination 808
Predication with Vertical Combination 808
End of Thread 808
Assigning Conditional Flags 809
Destination Hazard 812
Non-present Operands 812
Instruction Prefetch 813
ISA Introduction 813
Introducing the Execution Unit 814
EU Terms and Acronyms 816
Execution Units (EUs) 821
EU Changes by Processor Generation 821
EU Notation 824
Execution Environment 825
EU Data Types 825
Fundamental Data Types 825
Numeric Data Types 826

Integer Numeric Data Types 826
Floating-Point Numeric Data Types 828
Packed Signed Half-Byte Integer Vector 830
Packed UnSigned Half-Byte Integer Vector 831
Packed Restricted Float Vector 832
Floating Point Modes 834
IEEE Floating Point Mode 834
Partial Listing of Honored IEEE-754 Rules 834
Complete Listing of Deviations or Additional Requirements vs IEEE-754 835
Min Max of Floating Point Numbers 836
Alternative Floating Point Mode 837
Floating-Point Support 839
Floating-Point Types and Values 839
Not a Number (NaN) Formats 840
Floating-Point Rounding Modes 841
Floating-Point Operations and Precision 842
Single Precision Floating-Point Rounding to Integral Values 842
Floating-Point to Integer Conversion 842
Integer to Floating-Point Conversion 843
Floating-Point Min/Max Operations 844
IEEE Floating-Point Exceptions 844
Signaling Floating-Point Exceptions 845
Invalid Operation Exception 846
Division by Zero Exception 847
Overflow Exception 847
Underflow Exception 848
Inexact Exception 849
Floating-Point Compare Operations 849
Type Conversion 854
Float to Integer 854
Integer to Integer with Same or Higher Precision 854
Integer to Integer with Lower Precision 855
Integer to Float 855

Double Precision Float to Single Precision Float 855
Single Precision Float to Double Precision Float 855
Exceptions 856
Exception-Related Architecture Registers 857
System Routine 858
Invoking the System Routine 858
Returning to the Application Thread 859
System IP (SIP) 860
System Routine Register Space 860
System Scratch Memory Space 861
Conditional Instructions Within the System Routine 861
Use of NoDDClr 862
Exception Descriptions 863
Illegal Opcode 863
Undefined Opcodes 863
Software Exception 863
Context Save and Restore 863
Events That Do Not Generate Exceptions 864
Illegal Instruction Format 864
Malformed Message 864
GRF Register Out of Bounds 864
Hung Thread 865
Instruction Fetch Out of Bounds 865
FPU Math Errors 865
Computational Overflow 865
System Routine Example 865
Instruction Set Summary 868
Instruction Set Characteristics 868
SIMD Instructions and SIMD Width 868
Instruction Operands and Register Regions 868
Instruction Execution 869
Instruction Formats 869
Instruction Fields 873

Native Instruction Layouts 887
EU Compact Instructions 892
Opcode Encoding 892
Move and Logic Instructions 892
Flow Control Instructions 893
Miscellaneous Instructions 894
Parallel Arithmetic Instructions 894
Vector Arithmetic Instructions 895
Special Instructions 896
Native Instruction BNF 896
Instruction Groups 896
Destination Register 898
Source Register 899
Address Registers 900
Register Files and Register Numbers 900
Relative Location and Stack Control 902
Regions 902
Types 902
Write Mask 902
Swizzle Control 903
Immediate Values 903
Predication and Modifiers 903
Instruction Options 904
Instruction Set Summary Tables 905
Accumulator Restrictions 907
Instruction Set Reference 910
Conventions 910
Pseudo Code Format 910
General Macros and Definitions 911
Evaluate Write Enable 911
EUISA Instructions 912
Round Instructions 914
rndd – Round Down 914

rnde – Round to Nearest or Even 915
rndu – Round Up 916
rndz – Round to Zero 917
math – Extended Math Function 918
INV - Inverse 918
LOG – Logarithm 919
EXP - Exponent 919
SQRT - Square Root 919
RSQ - Reciprocal Square Root 920
POW - Power Function 920
SIN - SINE 921
COS - COSINE 922
INT DIV - Integer Divide 922
INVM/RSQRTM 923
Send Message 925
Send Message 925
EUISA Structures 929
EUISA Enumerations 931
EU Programming Guide 932
Assembler Pragmas 932
Declarations 932
Defaults and Defines 932
Example Pragma Usages 933
Assembly Programming Guideline 935
Usage Examples 936
Vector Immediate 936
Supporting DirectX 10 Pixel Shader Indexing 936
Supporting OpenGL Vertex Shader Instruction SWZ 937
Destination Mask for DP4 and Destination Dependency Control 938
Null Register as the Destination 939
Use of LINE Instruction 939
Mask for SEND Instruction 940
Channel Enables for Extended Math Unit 940

Channel Enables for Scratch Memory 942
Flow Control Instructions 944
Execution Masking 945
Branching 945
Fast-If 945
Cascade Branching 946
Compound Branches 947
Looping 948
Indexed Jump 952

3D-Compute Front End and Software Interface

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. Refer to each registers description and related feature for more information on each individual bit.

The registers detailed in this chapter are used across the SNB+ family of products and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

Predication

Predicate Render Registers

Register
MI_PREDICATE_SRC0 - Predicate Rendering Temporary Register0
MI_PREDICATE_SRC1 - Predicate Rendering Temporary Register1
MI_PREDICATE_DATA - Predicate Rendering Data Storage
MI_PREDICATE_RESULT - Predicate Rendering Data Result
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

MI_SET_PREDICATE

MI_SET_PREDICATE is a command that allows the driver to conditionally execute or skip a command during execution time, as detailed in the instruction definition:

The following is a list of commands that can be programmed when the PREDICATE ENABLE field in MI_SET_PREDICATE allows predication. Commands not listed here will have undefined behavior when executed with predication enabled:

Command
3DSTATE_URB_VS
3DSTATE_URB_HS
3DSTATE_URB_DS
3DSTATE_URB_GS
3DSTATE_PUSH_CONSTANT_ALLOC_VS
3DSTATE_PUSH_CONSTANT_ALLOC_HS
3DSTATE_PUSH_CONSTANT_ALLOC_DS
3DSTATE_PUSH_CONSTANT_ALLOC_GS
3DSTATE_PUSH_CONSTANT_ALLOC_PS
MI_LOAD_REGISTER_IMM
3DSTATE_WM_HZ_OP

Command
MEDIA_VFE_STATE
MEDIA_OBJECT
MEDIA_OBJECT_WALKER
MEDIA_INTERFACE_DESCRIPTOR_LOAD

MI_PREDICATE

The MI_PREDICATE command is used to control the Predicate state bit, which in turn can be used to enable/disable the processing of 3DPRIMITIVE commands.

MI_PREDICATE

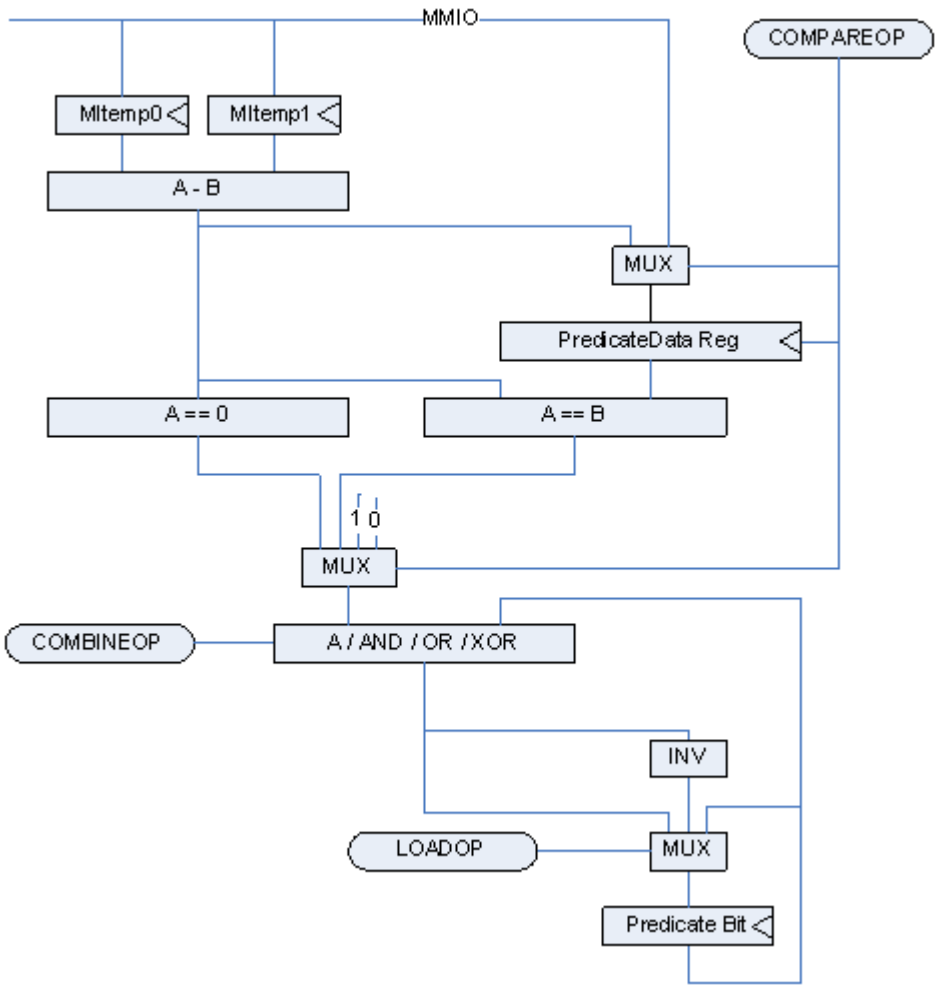
Predicated Rendering Support in HW

DX10 defines predicated rendering, where sequences of rendering commands can be discarded based on the result of a previous predicate test. A new state bit, Predicate, has been added to the command stream. In addition, a PredicateEnable bit is added to 3DPRIMITIVE. When the PredicateEnable bit is set, the command is ignored if the Predicate state bit is set.

A new command, MI_PREDICATE, is added. It contains several control fields which specify how the Predicate bit is generated.

Refer to the diagram below and the command description (linked above) for details.

MI_PREDICATE Function



MI_LOAD_REGISTER_MEM commands can be used to load the MItemp0, MItemp1, and PredicateData registers prior to MI_PREDICATE. To ensure the memory sources of the MI_LOAD_REGISTER_MEM commands are coherent with previous 3D_PIPECONTROL store-DWord operations, software can use the new Pipe Control Flush Enable bit in the PIPE_CONTROL command.

3D-Compute Front End and Software Interface

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. Refer to each registers description and related feature for more information on each individual bit.

The registers detailed in this chapter are used across the SNB+ family of products and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

VF Instance Count Registers

VF Instance Count Register Set		
Register Type:	MMIO_VF	
Address:	08300h - 08384h	
Default Value:	0000 0000h	
Access:	RO	
Size:	1088 bits	
Description:	Set of Registers for storing the index count values. In case of preempted drawcalls, these register store index count/number per element. For the non-preempted drawcalls, the values stored are ignored upon restore. These are saved as part of render context.	
DWord	**Bits**	**Description**
0	31:0	**Index Count 0.** Index Count value for Element 0. Format: U32
1	31:0	**Index Count 1.** Index Count value for Element 1. Format: U32
...	31:0	...
33	31:0	**Index Count 33.** Index Count value for Element 33. Format: U32

Registers
GAM_CTX - GAM Context Save Register

Mode and Misc Ctrl Registers

This section contains various registers for controls and modes

Controls/Modes
MI_MODE - Mode Register for Software Interface
GFX_MODE - Graphics Mode Register
GT_MODE - GT Mode Register
SAMPLER_MODE - SAMPLER Mode Register
CACHE_MODE_0 - Cache Mode Register 0
CACHE_MODE_1 - Cache Mode Register 1
GAFS_MODE - Mode Register for GAFS
FBC_RT_BASE_ADDR_REGISTER - FBC_RT_BASE_ADDR_REGISTER
FBC_RT_BASE_ADDR_REGISTER_UPPER - FBC_RT_BASE_ADDR_REGISTER_UPPER

L3CNTLREG - L3 Control Register

B/D/F/Type:

Address Offset: 0x7034

Default Value: 60000060h

Access: RW; RO;

Size: 32 bit

Below Register provides GT2 based L3 sizes.

For GT1 – all sizes need to be multiplied by 0.5.

For GT3 – all sizes need to be multiplied by 2.

For GT4 – all sizes need to be multiplied by 3.

All L3 ways have to be included in the programming to ensure that no ways are left out.

L3CNTLREG - L3 Control Register

Pipelines Statistics Counter Registers

These registers keep continuous count of statistics regarding the 3D pipeline. They are saved and restored with context but should not be changed by software except to reset them to 0 at context creation time. Write access to the statistics counter in this section must be done through MI_LOAD_REGISTER_IMM, MI_LOAD_REGISTER_MEM, or MI_LOAD_REGISTER_REG commands in ring buffer or batch buffer. These registers may be read at any time; however, to obtain a meaningful result, a pipeline flush just prior to reading the registers is necessary to synchronize the counts with the primitive stream.

Registers
IA_VERTICES_COUNT - IA Vertices Count
IA_PRIMITIVES_COUNT - Primitives Generated By VF
VS_INVOCATION_COUNT - VS Invocation Counter
HS_INVOCATION_COUNT - HS Invocation Counter
DS_INVOCATION_COUNT - DS Invocation Counter
GS_INVOCATION_COUNT - GS Invocation Counter
GS_PRIMITIVES_COUNT - GS Primitives Counter
CL_INVOCATION_COUNT - Clipper Invocation Counter
PS_INVOCATION_COUNT - PS Invocation Count
PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0
PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1
PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2
PS_DEPTH_COUNT
PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0
PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1
PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2
PS_DEPTH_COUNT_SLICE3 - PS Depth Count for Slice3
TIMESTAMP - Reported Timestamp Count
Stream Output Num Primitives Written Counter
Stream Output Write Offsets
Window Hardware Generated Clear Value

CS_CTX_TIMESTAMP- CS Context Timestamp Count:

This register provides a mechanism to obtain cumulative run time of a GPU context on HW.

Register
CS_CTX_TIMESTAMP - CS Context Timestamp Count

Diagram below details on when CS_CTX_TIMESTAMP reun time, save/restored during a GPGPU context switch flow.

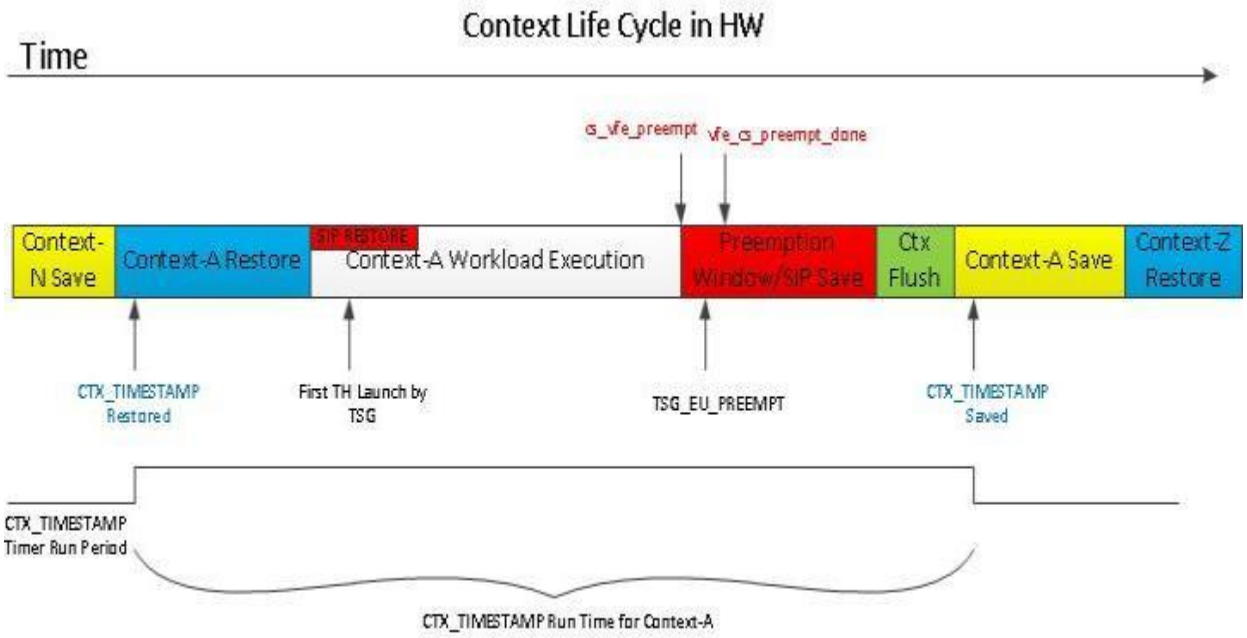


Fig: CTX_TIMESTAMP fucntionality during context execution

AUTO_DRAW Registers

3DPRIM_END_OFFSET - Auto Draw End Offset

3DPRIM_START_VERTEX - Load Indirect Start Vertex

3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count

3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count

3DPRIM_START_INSTANCE - Load Indirect Start Instance

3DPRIM_BASE_VERTEX - Load Indirect Base Vertex

MMIO Registers for GPGPU Indirect Dispatch

These registers are normally written with the MI_LOAD_REGISTER_MEMORY command rather than from the CPU.

GPGPU_DISPATCHDIMX - GPGPU Dispatch Dimension X

GPGPU_DISPATCHDIMY - GPGPU Dispatch Dimension Y

GPGPU_DISPATCHDIMZ - GPGPU Dispatch Dimension Z

TS_GPGPU_THREADS_DISPATCHED - Count Active Channels Dispatched

CS ALU Programming and Design

Command streamer implements a rudimentary ALU which supports basic Arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64bit operands. ALU has two 64bit registers at the input SRCA and SRCB to which the operands should be loaded on which operations will be performed and outputted to a 64 bit Accumulator. Zero Flag and Carry Flag are set based on accumulator output.

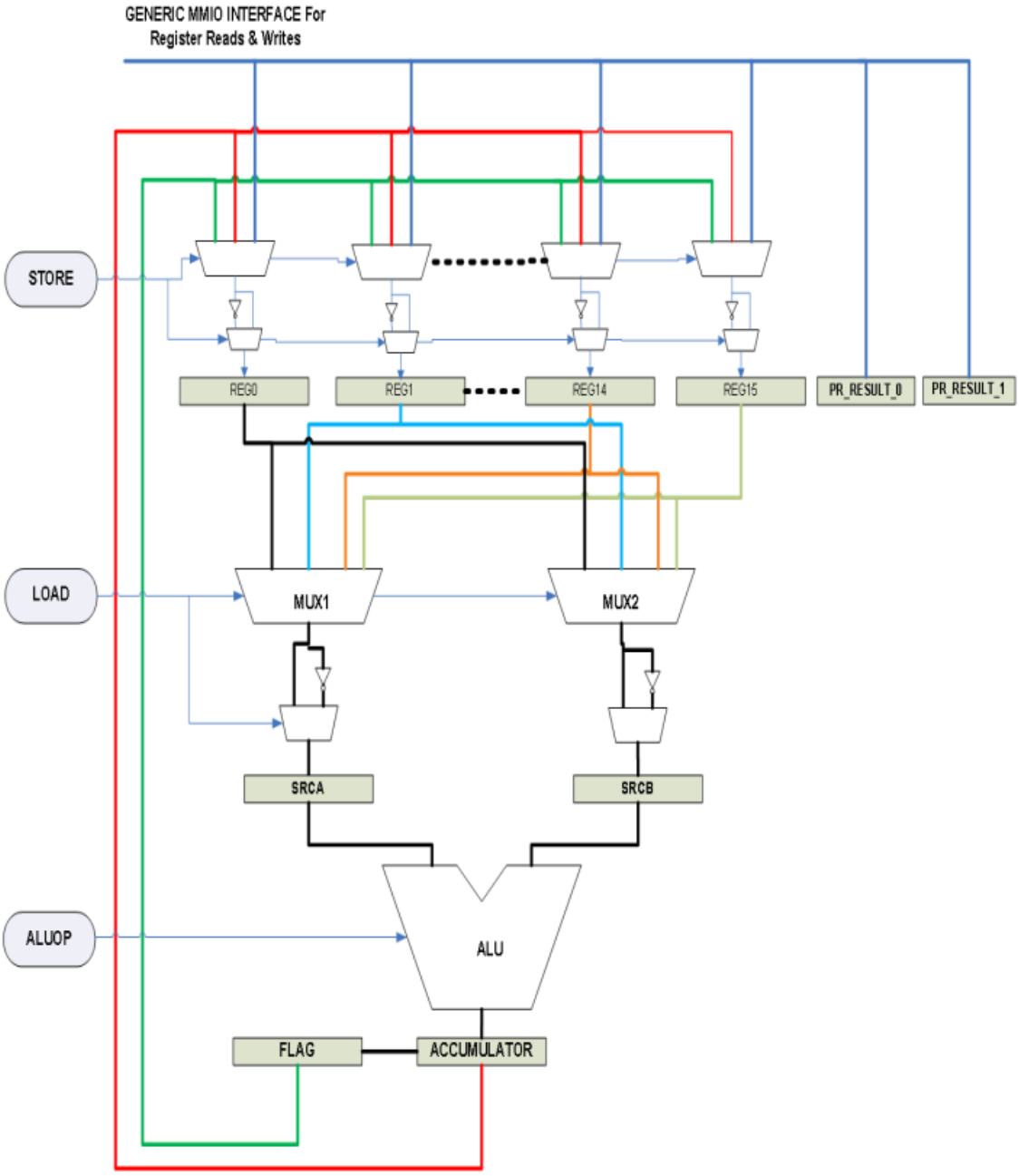
CS ALU Programming and Design

Command streamer implements a rudimentary ALU which supports basic Arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64bit operands. ALU has two 64bit registers at the input SRCA and SRCB to which the operands should be loaded on which operations will be performed and outputted to a 64 bit Accumulator. Zero Flag and Carry Flag are set based on accumulator output.

Generic Purpose Registers

Command streamer implements sixteen 64 bit General Purpose Registers which are MMIO mapped. These registers can be accessed similar to any other MMIO mapped registers through LRI, SRM, LRR, LRM or CPU access path for reads and writes. These registers will be labeled as R0, R1, … R15 throughout the discussion. Refer table in the B-spec update section mapping these registers to corresponding MMIO offset. A selected GPR register can be moved to SRCA or SRCB register using “LOAD” instruction. Outputs of the ALU, Accumulator, ZF and CF can be moved to any of the GPR using “STORE” instruction.

ALU BLOCK Diagram



Instruction Set

The instructions supported by the ALU can be broadly categorized into three groups:

- To move data from GPR to SRCA/SRCB – LOAD instruction.
- To move data from ACCUMULATOR/CF/ZF to GPR – STORE Instruction.

- To do arithmetic/Logical operations on SRCA and SRCB of ALU - ADD/SUB/AND/XOR/OR. Note: Accumulator is loaded with value of SRCA - SRCB on a subtraction.

Instruction Format

Each instruction is one Dword in size and consists of an ALU OPCODE, OPERAND1 and OPERAND2 in the format shown below.

ALU OPCODE	Operand-1	Operand-2
12 bits	10 bits	10 bits

LOAD Operation

The LOAD instruction moves the content of the destination register (Operand2) into the source register (Operand1). The destination register can be any of the GPR (R0, R1, ..., R15) and the source registers are SRCA and SRCB of the ALU. This is the only means SRCA and SRCB can be programmed.

LOAD has different flavors, wherein one can load the inverted version of the source register into the destination register or a hard coded value of all Zeros and All ones.

```
 // Loads any of Reg0 to Reg15 into the SRCA or SRCB registers of ALU.

     LOAD <SRCA, SRCB>, <REG0..REG15>

 // Loads inverted (bit wise) value of the mentioned Reg0 to 15 into SRCA or SRCB registers of
ALU.

     LOADINV <SRCA, SRCB>, <REG0..REG15>

 // Loads "0" into SRCA or SRCB

     LOAD0 <SRCA, SRCB>

 // Loads "1" into SRCA or SRCB

     LOAD1 <SRCA, SRCB>
```

31	20	19	10	9	0
Opcode		**Operand1**		**Operand2**	
LOAD		SRCA/SRCB		R0,R1..R15	
LOADINV		SRCA/SRCB		R0,R1..R15	
LOAD0		SRCA/SRCB		N/A	
LOAD1		SRCA/SRCB		N/A	

Arithmetic/Logical Operations

ADD, SUB, AND, OR, and XOR are the Arithmetic and Logical operations supported by Arithmetic Logic Unit (ALU). When opcode corresponding to a logical operation is performed on SRCA and SRCB, the

result is sent to ACCUMULATOR (ACCU), CF and ZF. Note that ACCU is 64-bit register. A NOOP when submitted to the ALU doesn't do anything, it is meant for creating bubble or kill cycles.

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
ADD	N/A	N/A
SUB	N/A	N/A
AND	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A
NOOP	N/A	NA

STORE Operation

The STORE instruction moves the content of the destination register (Operand1) into the source register (Operand2). The destination register can be accumulator (ACCU), CF or ZF. STORE has different flavors, wherein one can load the inverted version of the source register into destination register via STOREINV. When CF or ZF are stored, the same value is replicated on all 64 bits.

```
// Loads ACCMULATOR or Carry Flag or Zero Flag in to any of the generic registers
// Reg0 to Reg16. In case of CF and ZF same value is replicated on all the 64 bits.

    STORE   <R0.. R15>, <ACCU, CF, ZF >

// Loads inverted (ACCMULATOR or Carry Flag or Zero Flag) in to any of the
// generic registers Reg0 to Reg15.

    STOREINV <R0.. R15>, <ACCU, CF, ZF>
```

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
STORE	R0,R1..R15	ACCU/ZF/CF
STOREINV	R0, R1.. R15	ACCU/ZF/CF

Summary for ALU

Total Opcodes Supported: 12

Total Addressable Registers as source or destination: 21

- 16 GPR (R0, R1 …R15)
- 1 ACCU
- 1ZF
- 1CF
- SRCA, SRCB

Summary of Instructions Supported

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
LOAD	SRCA/SRCB	REG0..REG15
LOADINV	SRCA/SRCB	REG0..REG15
LOAD0	SRCA/SRCB	N/A
LOAD1	SRCA/SRCB	N?A
ADD	N/A	N/A
SUB	N/A	N/A
AND	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A
NOOP	N/A	N/A
STORE	REG0..REG15	ACCU/CF/ZF
STOREINV	REG0..REG15	ACCU/CF/ZF

Table for ALU OPCODE Encodings

ALU OPCODE	OPCODE ENCODING
NOOP	0x000
LOAD	0x080
LOADINV	0x480
LOAD0	0x081
LOAD1	0x481
ADD	0x100
SUB	0x101
AND	0x102
OR	0x103
XOR	0x104
STORE	0x180
STOREINV	0x580

In the above mentioned table, ALU Opcode Encodings look like random numbers. The rationale behind those encodings is because the ALU Opcode is further broken down into sub-sections for ease-of-design implementation.

PREFIX		OPCODE		SUBOPCODE	
11	10	9	7	6	0
PREFIX VALUE		**Description**			
0		Regular			
1		Invert			
OPCODE VALUE		**Description**			
0		NOOP			
1		LOAD			
2		ALU			
3		STORE			

ALU OPCODE	ENCODING	PREFIX	OPCODE	SUBOPCODE
		10	9 7	6 0
NOOP	0x000	0	0	0
LOAD	0x080	0	1	0
LOADINV	0x480	1	1	0
LOAD0	0x081	0	1	1
LOAD1	0x481	1	1	1
ADD	0x100	0	2	0
SUB	0x101	0	2	1
AND	0x102	0	2	2
OR	0x103	0	2	3
XOR	0x104	0	2	4
STORE	0x180	0	3	0
STOREINV	0x580	1	3	0

Table for Register Encodings

Register	Register Encoding
R0	0x0
R1	0x1
R2	0x2
R3	0x3
R4	0x4
R5	0x5
R6	0x6
R7	0x7

Register	Register Encoding
R8	0x8
R9	0x9
R10	0xa
R11	0xb
R12	0xc
R13	0xd
R14	0xe
R15	0xf
SRCA	0x20
SRCB	0x21
ACCU	0x31
ZF	0x32
CF	0x33

CS_GPR - Command Streamer General Purpose Registers

Following are Command Streamer General Purpose Registers:

CS_GPR - General Purpose Register

Memory Interface Commands for Rendering Engine

Command
MI_SET_CONTEXT
MI_TOPOLOGY_FILTER
MI_URB_ATOMIC_ALLOC
MI_LOAD_URB_MEM
MI_STORE_URB_MEM

State Commands

This section covers the following commands:

- **STATE_PREFETCH command.** The STATE_PREFETCH command is provided strictly as an optional mechanism to possibly enhance pipeline performance by prefetching data into the GPE's Instruction and State Cache (ISC).
- **STATE_SIP command**

Command
STATE_SIP
3DSTATE_URB_CLEAR

STATE_BASE_ADDRESS

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE. (See Memory Access Indirection for details.)

The following commands must be reissued following any change to the base addresses:

- 3DSTATE_PIPELINE_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- MEDIA_STATE_POINTERS

Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

Command
STATE_BASE_ADDRESS
PIPELINE_SELECT

The **Pipeline Select** state is contained within the logical context.

Synchronization of the 3D Pipeline

Two types of synchronizations are supported for the 3D pipe: top of the pipe and end of the pipe. Top of the pipe synchronization really enforces the read-only cache invalidation. This synchronization guarantees that primitives rendered after such synchronization event fetches the latest read-only data from memory. End of the pipe synchronization enforces that the read and/or read-write buffers do not have outstanding hardware accesses. These are used to implement read and write fences as well as to write out certain statistics deterministically with respect to progress of primitives through the pipeline (and without requiring the pipeline to be flushed.) The PIPE_CONTROL command (see details below) is used to perform all of above synchronizations.

Top-of-Pipe Synchronization

Top-of-pipe synchronization refers to SW actions to prepare HW for new state-binding at the beginning of the rendering sequence in a given context. HW may have residual states cached in the state-caches and read-only surfaces in various caches. With new rendering sequence, read-only surfaces may go through change in the binding. Hence read-only invalidation is required before such new rendering sequence. Read-only cache invalidation is top-of-pipe synchronization. Upon parsing this specific pipe-control command, HW invalidates all caches in GT domain that have read-only surfaces but does not guarantee invalidation beyond GT caches (i.e. LLC). Further, HW does not guarantee that all prior accesses to those read-only surfaces have completed. Therefore SW must guarantee that there are no pending accesses to those read-only surfaces before initializing the top-of-pipe synchronization. PIPE-CONTROL command described below allows for invalidating individual read-only stream type. It is recommended that driver invalidates only the required caches on the need basis so that cache warm-up overhead can be reduced.

End-of-Pipe Synchronization

The driver can use end-of-pipe synchronization to know that rendering is complete (although not necessarily in memory) so that it can deallocate in-memory rendering state, read-only surfaces, instructions, and constant buffers. An end-of-pipe synchronization point is also sufficient to guarantee that all pending depth tests have completed so that the visible pixel count is complete prior to storing it to memory. End-of-pipe completion is sufficient (although not necessary) to guarantee that read events are complete (a “read fence” completion). Read events are still pending if work in the pipeline requires any type of read except a render target read (blend) to complete.

Write synchronization is a special case of end-of-pipe synchronization that requires that the render cache and/or depth related caches are flushed to memory, where the data will become globally visible. This type of synchronization is required prior to SW (CPU) actually reading the result data from memory, or initiating an operation that will use as a read surface (such as a texture surface) a previous render target and/or depth/stencil buffer. Exercising the write cache flush bits (Render Target Cache Flush Enable, Depth Cache Flush Enable, DC Flush) in PIPE_CONTROL only ensures the write caches are flushed and doesn’t guarantee the data is globally visible.

SW can track the completion of the end-of-pipe-synchronization by using “Notify Enable” and “Post-Sync Operation - Write Immediate Data” in the PIPE_CONTROL command. “Notify Enable” and “Post-Sync Operation - Write Immediate Data” generate a fence cycle on achieving end-of-pipe-synchronization for the corresponding PIPE_CONTROL command. Fence cycle ensures all the write cycles in front of it are to global visible point before they themselves get processed.It is guaranteed the data flushed out by the PIPE_CONTROL is updated in memory by the time SW receives the corresponding Pipe Control Notify interrupt.

In case the data flushed out by the render engine is to be read back in to the render engine in coherent manner, then the render engine has to wait for the fence completion before accessing the flushed data. This can be achieved by following means on various products:

PIPE_CONTROL command with CS Stall and the required write caches flushed with Post-Sync-Operation as Write Immediate Data.

Example:

- WorkLoad-1 (3D/GPGPU/MEDIA)
- PIPE_CONTROL (CS Stall, Post-Sync-Operation Write Immediate Data, Required Write Cache Flush bits set)
- WorkLoad-2 (Can use the data produced or output by Workload-1)

Synchronization Actions

In order for the driver to act based on a synchronization point (usually the whole point), the reaching of the synchronization point must be communicated to the driver. This section describes the actions that may be taken upon completion of a synchronization point which can achieve this communication.

Writing a Value to Memory

The most common action to perform upon reaching a synchronization point is to write a value out to memory. An immediate value (included with the synchronization command) may be written. In lieu of an immediate value, the 64-bit value of the PS_DEPTH_COUNT (visible pixel count) or TIMESTAMP register may be written out to memory. The captured value will be the value at the moment all primitives parsed prior to the synchronization commands have been completely rendered, and optionally after all said primitives have been pushed to memory. It is not required that a value be written to memory by the synchronization command.

Visible pixel or TIMESTAMP information is only useful as a delta between 2 values, because these counters are free-running and are not to be reset except at initialization. To obtain the delta, two PIPE_CONTROL commands should be initiated with the command sequence to be measured between them. The resulting pair of values in memory can then be subtracted to obtain a meaningful statistic about the command sequence.

PS_DEPTH_COUNT

If the selected operation is to write the visible pixel count (PS_DEPTH_COUNT register), the synchronization command should include the **Depth Stall Enable** parameter. There is more than one point at which the global visible pixel count can be affected by the pipeline; once the synchronization command reaches the first point at which the count can be affected, any primitives following it are stalled at that point in the pipeline. This prevents the subsequent primitives from affecting the visible pixel count until all primitives preceding the synchronization point reach the end of the pipeline, the visible pixel count is accurate and the synchronization is completed. This stall has a minor effect on performance and should only be used in order to obtain accurate “visible pixel” counts for a sequence of primitives.

The PS_DEPTH_COUNT count can be used to implement an (API/DDI) “Occlusion Query” function.

Generating an Interrupt

The synchronization command may indicate that a "Sync Completion" interrupt is to be generated (if enabled by the MI Interrupt Control Registers – see *Memory Interface Registers*) once the rendering of all prior primitives is complete. Again, the completion of rendering can be considered to be when the internal render cache has been updated, or when the cache contents are visible in memory, as selected by the command options.

Invalidating of Caches

If software wishes to use the notification that a synchronization point has been reached in order to reuse referenced structures (surfaces, state, or instructions), it is not sufficient just to make sure rendering is complete. If additional primitives are initiated after new data is laid over the top of old in memory following a synchronization point, it is possible that stale cached data will be referenced for the subsequent rendering operation. In order to avoid this, the PIPE_CONTROL command must be used. (See PIPE_CONTROL Command description).

PIPE_CONTROL Command

The PIPE_CONTROL command provides mechanism to achieve the synchronization of the 3D pipeline and to execute post-synchronization operations as described in the section "Synchronization of the 3D pipeline". Parsing a PIPE_CONTROL command stalls the 3D pipe only if the stall enable bit is set. Commands after PIPE_CONTROL will continue to be parsed and processed in the 3D pipeline. This may include additional PIPE_CONTROL commands. The implementation does enforce a practical upper limit (8) on the number of PIPE_CONTROL commands that may be outstanding at once. Parsing a PIPE_CONTROL command that causes this limit to be reached will stall the parsing of new commands until the first of the outstanding PIPE_CONTROL commands reaches the end of the pipe and retires.

Although PIPE_CONTROL is intended for use with the 3D pipe, it is legal to issue PIPE_CONTROL when the Media pipe is selected. In this case PIPE_CONTROL will stall at the top of the pipe until the Media FFs finish processing commands parsed before PIPE_CONTROL. Post-synchronization operations, flushing of caches and interrupts will then occur if enabled via PIPE_CONTROL parameters. Due to this stalling behavior, only one PIPE_CONTROL command can be outstanding at a time on the Media pipe.

For the invalidate operation of the pipe control, the following pointers are affected. The invalidate operation affects the restore of these packets. If the pipe control invalidate operation is completed before the context save, the indirect pointers will not be restored from memory.

- Pipeline State Pointer
- Media State Pointer
- Constant Buffer Packet

It is up to software to program the appropriate read-only cache invalidation such as the sampler and constant read caches or the instruction and state caches. Once notification is observed, new data may then be loaded (potentially "on top of" the old data) without fear of stale cache data being referenced for subsequent rendering.

If software wishes to access the rendered data in memory (for analysis by the application or to copy it to a new location to use as a texture, for example), it must also ensure that the write cache (render cache) is flushed after the synchronization point is reached so that memory will be updated. This can be done by setting the **Write Cache Flush Enable** bit. Note that the **Depth Stall Enable** bit must be clear in order for the flush of the render cache to occur. **Depth Stall Enable** is intended only for accurate reporting of the PS_DEPTH counter; the render cache cannot be flushed nor can the read caches be invalidated (except for the instruction/state cache) in conjunction with this operation.

Vertex caches are only invalidated when the VF invalidate bit is set in PIPE_CONTROL (i.e. decision is done in software, not hardware) Note that the index-based vertex cache is always flushed between primitive topologies and of course PIPE_CONTROL can only be issued between primitive topologies. Therefore only the VF ("address-based") cache is uniquely affected by PIPE_CONTROL.

PIPE_CONTROL

PIPE_CONTROL

Description
Hardware supports up to 32 pending PIPE_CONTROL flushes.

The table below explains all the different flush/invalidation scenarios.

Caches Invalidated/Flushed by PIPE_CONTROL Bit Settings

Write Cache Flush	Notification Enabled	Non-VF RO Cache Invalidate	VF RO Cache Invalidate	Marker Sent	Pipeline Marker Enable	Completion Requested	Top of Pipe Invalidate Pulse from CS
0	0	0	0	N/A	N/A	N/A	N/A
0	0	0	1	Yes	No	N/A	No
0	0	1	0	No	N/A	N/A	Yes
0	0	1	1	Yes	No	No	Yes
X	1	0	X	Yes	Yes	Yes	No
X	1	1	X	Yes	Yes	Yes	Yes
1	X	0	X	Yes	Yes	Yes	No
1	X	1	X	Yes	Yes	Yes	Yes

Programming Restrictions for PIPE_CONTROL

PIPE_CONTROL arguments can be split up into three categories:

- Post-sync operations
- Flush Types
- Stall

Post-sync operation is only indirectly affected by the flush type category via the stall bit. The stall category depends on the both flush type and post-sync operation arguments. A PIPE_CONTROL with no arguments set is **Invalid**.

Post-Sync Operation

These arguments relate to events that occur after the marker initiated by the PIPE_CONTROL command is completed. The table below shows the restrictions:

Argument	Bits	Restriction
LRI Post Sync Operation	23	Post Sync Operation ([15:14] of DW1) must be set to 0x0.
Protected Mem Enable	22	Requires stall bit ([20] of DW1) set.
Global Snapshot Count Reset	19	This bit must not be exercised on any product. Requires stall bit ([20] of DW1) set.
Generic Media State Clear	16	Requires stall bit ([20] of DW1) set.
Indirect State Pointers Disable	9	Requires stall bit ([20] of DW1) set.
Store Data Index	21	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0'.
Sync GFDT	17	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0' or 0x2520[13] must be set.
TLB inv	18	Requires stall bit ([20] of DW1) set.
Post Sync Op	15:14	LRI Post Sync Operation ([23] of DW1) must be set to '0'.
Protected Memory Application ID	6	Requires stall bit ([20] of DW1) set.

Flush Types

These are arguments related to the type of read only invalidation or write cache flushing is being requested. Note that there is only intra-dependency. That is, it is not affected by the post-sync operation or the stall bit. The table below shows the restrictions:

Arguments	Bit	Restrictions
Stall Pixel Scoreboard	1	No Restriction.
Inst invalidate	11	No Restriction.
Tex invalidate	10	Requires stall bit ([20] of DW) set for all GPGPU Workloads.
VF invalidate	4	“Post Sync Operation” must be enabled to “Write Immediate Data” or “Write PS Depth Count” or “Write Timestamp”.
Constant invalidate	3	No Restriction.
State Invalidate	2	No Restriction.

Stall

If the stall bit is set, the command streamer waits until the pipe is completely flushed.

Arguments	Bit	Restrictions
Stall Bit	20	No Restrictions.

Context Image

Logical Contexts are memory images used to store copies of the device's rendering and ring context.

Logical Contexts are aligned to 256-byte boundaries.

Logical contexts are referenced by their memory address. The format and contents of rendering contexts are considered ***device-dependent*** and software must not access the memory contents directly. The definition of the logical rendering and power context memory formats is included here primarily for internal documentation purposes.

Power Context Image

Render Engine Power Context

The table below captures the data from CS power context save/restored by PM. Address offsets in this table are relative to the starting location of CS in the overall power context image managed by PM.

RCS Power Context Image

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
NOOP		CS	1	0	CSFE
Load_Register_Immediate header	0x1100_108B	CS	1	0001	CSFE
GFX_MODE	0x229C	CS	2	0002	CSFE
GHWSP	0x2080	CS	2	0004	CSFE
RING_BUFFER_CONTROL (Ring Always Disabled)	0x203C	CS	2	0006	CSFE
Ring Head Pointer Register	0x2034	CS	2	0008	CSFE
Ring Tail Pointer Register	0x2030	CS	2	000A	CSFE
RING_BUFFER_START	0x2038	CS	2	000C	CSFE
RING_BUFFER_CONTROL (Original status)	0x203C	CS	2	000E	CSFE
Batch Buffer Current Head Register (UDW)	0x2168	CS	2	0010	CSFE
Batch Buffer Current Head Register	0x2140	CS	2	0012	CSFE
Batch Buffer State Register	0x2110	CS	2	0014	CSFE
SECOND_BB_ADDR_UDW	0x211C	CS	2	0016	CSFE
SECOND_BB_ADDR	0x2114	CS	2	0018	CSFE

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
SECOND_BB_STATE	0x2118	CS	2	001A	CSFE
RC_PSMI_CONTROL	0x2050	CS	2	001C	CSFE
RC_PWRCTX_MAXCNT	0x2054	CS	2	001E	CSFE
CTX_WA_PTR	0x2058	CS	2	0020	CSFE
NOPID	0x2094	CS	2	0022	CSFE
HWSTAM	0x2098	CS	2	0024	CSFE
IMR	0x20A8	CS	2	0026	CSFE
EIR	0x20B0	CS	2	0028	CSFE
EMR	0x20B4	CS	2	002A	CSFE
CMD_CCTL_0	0x20C4	CS	2	002C	CSFE
UHPTR	0x2134	CS	2	002E	CSFE
BB_PREEMPT_ADDR_UDW	0x216C	CS	2	0030	CSFE
BB_PREEMPT_ADDR	0x2148	CS	2	0032	CSFE
RING_BUFFER_HEAD_PREEMPT_REG	0x214C	CS	2	0034	CSFE
PREEMPT_DLY	0x2214	CS	2	0036	CSFE
CTXT_PREMP_DBG	0x2248	CS	2	0038	CSFE
SYNC_FLIP_STATUS	0x22D0	CS	2	003A	CSFE
SYNC_FLIP_STATUS_1	0x22D4	CS	2	003C	CSFE
SYNC_FLIP_STATUS_2	0x22EC	CS	2	003E	CSFE
WAIT_FOR_RC6_EXIT	0x20CC	CS	2	0040	CSFE
RCS_CTXID_PREEMPTION_HINT	0x24CC	CS	2	0042	CSFE
CS_PREEMPTION_HINT_UDW	0x24C8	CS	2	0044	CSFE
CS_PREEMPTION_HINT	0x24BC	CS	2	0046	CSFE
CCID Register	0x2180	CS	2	0048	CSFE
SBB_PREEMPT_ADDRESS_UDW	0x2138	CS	2	004A	CSFE
SBB_PREEMPT_ADDRESS	0x213C	CS	2	004C	CSFE
MI_PREDICATE_RESULT_2	0x23BC	CS	2	004E	CSFE
CTXT_ST_PTR	0x23A0	CS	2	0050	CSFE
CTXT_ST_BUF	0x2370	CS	24	0052	CSFE
SEMA_WAIT_POLL	0x224C	CS	2	006A	CSFE
IDLEDELAY	0x223C	CS	2	006C	CSFE
DISPLAY MESSAGE FORWARD STATUS	0x22E8	CS	2	006E	CSFE
RCS_FORCE_TO_NONPRIV	0x24D0	CS	24	0070	CSFE
EXECLIST_STATUS_REGISTER	0x2234	CS	2	0088	CSFE
CXT_OFFSET	0x21AC	CS	2	008C	CSBE

Description	Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
NOOP		CS	2	008E	CSFE
NOOP		CS	1	0090	CSBE
Load_Register_Immediate header	0x1100_1021	CS	1	0091	CSBE
FF_THREAD_MODE	0x20A0	CS	2	0096	CSBE
GAFS_Mode	0x212C	CS	2	009C	CSBE
CXT_SIZE	0x21A8	CS	2	009E	CSBE
RS_OFFSET	0x21B4	CS	2	00A4	CSBE
RS_PREEMPTION_HINT_UDW	0x24C4	CS	2	00A6	CSBE
RS_PREEMPTION_HINT	0x24C0	CS	2	00A8	CSBE
URB_CTX_OFFSET	0x21B8	CS	2	00AA	CSBE
VF_CTX_OFFSET	0x21BC	CS	2	00AC	CSBE
VF PREMPTION VERTEX HINT	0x83B0	VF	2	00B0	CSBE
VF PREEMPTION INSTANCE HINT	0x83B4	VF	2	00B2	CSBE
NOOP		CS	10	00B4	CSBE
NOOP		CS	1	00BE	CSBE
MI_BATCH_BUFFER_END		CS	1	00BF	CSBE

Engine Register and State Context

This section describes programming requirements for the Register State Context.

Programming Note	
Context:	Register State Context

- All the MMIO registers part of the “Engine Register and State Context Image” are context specific and gets context save/restored upon a context switch. MMIO register values belonging to a context can be exercised through HOST/IA MMIO interface only when the context is active in HW. Exercising context specific MMIO registers through HOST/IA MMIO is completely asynchronous to the context execution in HW and can’t guarantee a desired sampling point during execution. In execlist mode of scheduling there is no active context when HW is Idle.
- All the write access to MMIO registers listed in the “Engine Register and State Context image” subsections below must be done through MI commands (MI_LOAD_REGISTER_IMM, MI_LOAD_REG_MEM, MI_LOAD_REGISTER_REG) in the command sequence.

LRCA Context Image

EXECLIST CONTEXT(Ring)
EXECLIST CONTEXT (PPGTT Base)
ENGINE CONTEXT(CS)
ENGINE CONTEXT(GAMT)
ENGINE CONTEXT(LNCF)
ENGINE CONTEXT(SVG)
ENGINE CONTEXT(SVL)
ENGINE CONTEXT(TDL)
ENGINE CONTEXT(WM)
ENGINE CONTEXT(SC)
ENGINE CONTEXT(DM)
ENGINE CONTEXT(SOL)
ENGINE CONTEXT(VF)
ENGINE CONTEXT(VFE)
EXTENDED ENGINE CONTEXT(RS)
URB_ATOMIC CONTEXT(URB)

Register State Context

Context Color Codes Used

EXECLIST CONTEXT
EXECLIST CONTEXT (PPGTT Base)
ENGINE CONTEXT
EXTENDED ENGINE CONTEXT
URB_ATOMIC CONTEXT

Register Information

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
NOOP		CS	1	0
Load_Register_Immediate header	0x1100_101B	CS	1	0001
Context Control	0x2244	CS	2	0002
Ring Head Pointer Register	0x2034	CS	2	0004
Ring Tail Pointer Register	0x2030	CS	2	0006
RING_BUFFER_START	0x2038	CS	2	0008
RING_BUFFER_CONTROL	0x203C	CS	2	000A
Batch Buffer Current Head Register (UDW)	0x2168	CS	2	000C
Batch Buffer Current Head Register	0x2140	CS	2	000E
Batch Buffer State Register	0x2110	CS	2	0010
SECOND_BB_ADDR_UDW	0x211C	CS	2	0012
SECOND_BB_ADDR	0x2114	CS	2	0014
SECOND_BB_STATE	0x2118	CS	2	0016
BB_PER_CTX_PTR	0x21C0	CS	2	0018
RCS_INDIRECT_CTX	0x21C4	CS	2	**001A**
RCS_INDIRECT_CTX_OFFSET	0x21C8	CS	2	**001C**
NOOP		CS	2	001E
NOOP		CS	1	0020
Load_Register_Immediate header	0x1100_1011	CS	1	0021
CTX_TIMESTAMP	0x23A8	CS	2	0022
PDP3_UDW0x1100_1029	0x228C	CS	2	0024
PDP3_LDW	0x2288	CS	2	0026
PDP2_UDW	0x2284	CS	2	0028
PDP2_LDW	0x2280	CS	2	002A
PDP1_UDW	0x227C	CS	2	002C
PDP1_LDW	0x2278	CS	2	002E
PDP0_UDW	0x2274	CS	2	0030
PDP0_LDW	0x2270	CS	2	0032
NOOP		CS	12	0034
NOOP		CS	1	0040
Load_Register_Immediate header	0x1100_0001	CS	1	0041
R_PWR_CLK_STATE	0x20C8	CS	2	0042
GPGPU_CSR_BASE_ADDRESS		CS	3	0044

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
NOOP		CS	9	0047
NOOP		CS	1	0050
Load_Register_Immediate header	0x1100_1057	CS	1	0051
EXCC	0x2028	CS	2	0052
MI_MODE	0x209C	CS	2	0054
INSTPM	0x20C0	CS	2	0056
PR_CTR_CTL	0x2178	CS	2	0058
PR_CTR_THRSH	0x217C	CS	2	005A
TIMESTAMP Register (LSB)	0x2358	CS	2	005C
BB_START_ADDR_UDW	0x2170	CS	2	005E
BB_START_ADDR	0x2150	CS	2	0060
BB_ADD_DIFF	0x2154	CS	2	0062
BB_OFFSET	0x2158	CS	2	0064
MI_PREDICATE_RESULT_1	0x241C	CS	2	0066
CS_GPR (1-16)	0x2600	CS	64	0068
IPEHR	0x2068	CS	2	**00A8**
NOOP		CS	6	**00AA**
NOOP		CS	1	00B0
Load_Register_Immediate header	0x1100_1085	CS	1	00B1
CS_CONTEXT_STATUS1	0x2184	CS	2	00B4
IA_VERTICES_COUNT	0x2310	CS	4	00B6
IA_PRIMITIVES_COUNT	0x2318	CS	4	00BA
VS_INVOCATION_COUNT	0x2320	CS	4	00BE
HS_INVOCATION_COUNT	0x2300	CS	4	00C2
DS_INVOCATION_COUNT	0x2308	CS	4	00C6
GS_INVOCATION_COUNT	0x2328	CS	4	00CA
GS_PRIMITIVES_COUNT	0x2330	CS	4	00CE
CL_INVOCATION_COUNT	0x2338	CS	4	00D2
CL_PRIMITIVES_COUNT	0x2340	CS	4	00D6
PS_INVOCATION_COUNT_0	0x22C8	CS	4	00DA
PS_DEPTH_COUNT _0	0x22D8	CS	4	00DE
GPUGPU_DISPATCHDIMX	0x2500	CS	2	00E4
GPUGPU_DISPATCHDIMY	0x2504	CS	2	00E6
GPUGPU_DISPATCHDIMZ	0x2508	CS	2	00E8

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
MI_PREDICATE_SRC0	0x2400	CS	2	00EA
MI_PREDICATE_SRC0	0x2404	CS	2	00EC
MI_PREDICATE_SRC1	0x2408	CS	2	00EE
MI_PREDICATE_SRC1	0x240C	CS	2	00F0
MI_PREDICATE_DATA	0x2410	CS	2	00F2
MI_PREDICATE_DATA	0x2414	CS	2	00F4
MI_PRED_RESULT	0x2418	CS	2	00F6
3DPRIM_END_OFFSET	0x2420	CS	2	00F8
3DPRIM_START_VERTEX	0x2430	CS	2	00FA
3DPRIM_VERTEX_COUNT	0x2434	CS	2	00FC
3DPRIM_INSTANCE_COUNT	0x2438	CS	2	00FE
3DPRIM_START_INSTANCE	0x243C	CS	2	0100
3DPRIM_BASE_VERTEX	0x2440	CS	2	0102
GPGPU_THREADS_DISPATCHED	0x2290	CS	4	0104
PS_INVOCATION_COUNT_1	0x22F0	CS	4	0108
PS_DEPTH_COUNT _1	0x22F8	CS	4	010C
RS_PREEMPT_STATUS	0x215C	CS	2	0110
CTX_SEMA_REG	0x24B4	CS	4	0112
PRODUCE_COUNT_BTP	0x2480	CS	2	0116
PRODUCE_COUNT_DX9_CONSTANTS	0x2484	CS	2	0118
PRODUCE_COUNT_GATHER_CONSTANTS	0x248C	CS	2	011A
PARSED_COUNT_BTP	0x2490	CS	2	011C
PARSED_COUNT_DX9_CONSTANTS	0x2494	CS	2	011E
OA_CTX_CONTROL	0x2360	CS	2	0128
OACTXID	0x2364	CS	2	012A
PS_INVOCATION_COUNT_2	0x2448	CS	4	012C
PS_DEPTH_COUNT_2	0x2450	CS	4	0130
RS_PREEMPT_STATUS_UDW	0x2174	CS	2	0134
NOOP		CS	8	0138
MI_TOPOLOGY_FILTER		CS	1	0140
NOOP		CS	2	0141
PIPELINE_SELECT		CS	1	0143
STATE_BASE_ADDRESS		CS	19	0144
3DSTATE_PUSH_CONSTANT_ALLOC_VS		CS	2	0157

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
3DSTATE_PUSH_CONSTANT_ALLOC_HS		CS	2	0159
3DSTATE_PUSH_CONSTANT_ALLOC_DS		CS	2	015B
3DSTATE_PUSH_CONSTANT_ALLOC_GS		CS	2	015D
3DSTATE_PUSH_CONSTANT_ALLOC_PS		CS	2	015F
3DSTATE_BINDING_TABLE_POOL_ALLOC		CS	4	0161
3DSTATE_GATHER_POOL_ALLOC		CS	4	0165
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC		CS	4	0169
MI_RS_CONTROL		CS	1	016D
MI_URB_ATOMIC_ALLOC		CS	1	016E
NOOP		CS	17	016F
NOOP		GAMT	1	0180
Load_Register_Immediate header	0x1100_108B	GAMT	1	0181
GFX_MOCS_0	C800	GAMT	2	0182
GFX_MOCS_1	C804	GAMT	2	0184
GFX_MOCS_2	C808	GAMT	2	0186
GFX_MOCS_3	C80C	GAMT	2	0188
GFX_MOCS_4	C810	GAMT	2	018A
GFX_MOCS_5	C814	GAMT	2	018C
GFX_MOCS_6	C818	GAMT	2	018E
GFX_MOCS_7	C81C	GAMT	2	0190
GFX_MOCS_8	C820	GAMT	2	0192
GFX_MOCS_9	C824	GAMT	2	0194
GFX_MOCS_10	C828	GAMT	2	0196
GFX_MOCS_11	C82C	GAMT	2	0198
GFX_MOCS_12	C830	GAMT	2	019A
GFX_MOCS_13	C834	GAMT	2	019C
GFX_MOCS_14	C838	GAMT	2	019E
GFX_MOCS_15	C83C	GAMT	2	01A0
GFX_MOCS_16	C840	GAMT	2	01A2
GFX_MOCS_17	C844	GAMT	2	01A4
GFX_MOCS_18	C848	GAMT	2	01A6
GFX_MOCS_19	C84C	GAMT	2	01A8
GFX_MOCS_20	C850	GAMT	2	01AA
GFX_MOCS_21	C854	GAMT	2	01AC

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
GFX_MOCS_22	C858	GAMT	2	01AE
GFX_MOCS_23	C85C	GAMT	2	01B0
GFX_MOCS_24	C860	GAMT	2	01B2
GFX_MOCS_25	C864	GAMT	2	01B4
GFX_MOCS_26	C868	GAMT	2	01B6
GFX_MOCS_27	C86C	GAMT	2	01B8
GFX_MOCS_28	C870	GAMT	2	01BA
GFX_MOCS_29	C874	GAMT	2	01BC
GFX_MOCS_30	C878	GAMT	2	01BE
GFX_MOCS_31	C87C	GAMT	2	01C0
GFX_MOCS_32	C880	GAMT	2	01C2
GFX_MOCS_33	C884	GAMT	2	01C4
GFX_MOCS_34	C888	GAMT	2	01C6
GFX_MOCS_35	C88C	GAMT	2	01C8
GFX_MOCS_36	C890	GAMT	2	01CA
GFX_MOCS_37	C894	GAMT	2	01CC
GFX_MOCS_38	C898	GAMT	2	01CE
GFX_MOCS_39	C89C	GAMT	2	01D0
GFX_MOCS_40	C8A0	GAMT	2	01D2
GFX_MOCS_41	C8A4	GAMT	2	01D4
GFX_MOCS_42	C8A8	GAMT	2	01D6
GFX_MOCS_43	C8AC	GAMT	2	01D8
GFX_MOCS_44	C8B0	GAMT	2	01DA
GFX_MOCS_45	C8B4	GAMT	2	01DC
GFX_MOCS_46	C8B8	GAMT	2	01DE
GFX_MOCS_47	C8BC	GAMT	2	01E0
GFX_MOCS_48	C8C0	GAMT	2	01E2
GFX_MOCS_49	C8C4	GAMT	2	01E4
GFX_MOCS_50	C8C8	GAMT	2	01E6
GFX_MOCS_51	C8CC	GAMT	2	01E8
GFX_MOCS_52	C8D0	GAMT	2	01EA
GFX_MOCS_53	C8D4	GAMT	2	01EC
GFX_MOCS_54	C8D8	GAMT	2	01EE
GFX_MOCS_55	C8DC	GAMT	2	01F0

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
GFX_MOCS_56	C8E0	GAMT	2	01F2
GFX_MOCS_57	C8E4	GAMT	2	01F4
GFX_MOCS_58	C8E8	GAMT	2	01F6
GFX_MOCS_59	C8EC	GAMT	2	01F8
GFX_MOCS_60	C8F0	GAMT	2	01FA
GFX_MOCS_61	C8F4	GAMT	2	01FC
GFX_MOCS_62	C8F8	GAMT	2	01FE
GFX_MOCS_63	C8FC	GAMT	2	0200
Tiled Resources VA Translation Table L3 Pointer	4DE0	GAMT	4	0202
Tiled Resources Null Tile Detection Register	4DE8	GAMT	2	0206
Tiled Resources Invalid Tile Detection Register	4DEC	GAMT	2	0208
Tiled Resources Virtual Address Detection Registers	4DF0	GAMT	2	020A
Tiled Resources Translation Table Control Register	4DF4	GAMT	2	020C
NOOP		GAM	2	020E
NOOP		LNCF	1	0210
Load_Register_Immediate header	0x1100_1001	LNCF	1	0211
L3CNTLREG	7034	LNCF	2	0212
NOOP		LNCF	1	0214
Load_Register_Immediate header	0x1100_103F	LNCF	1	0215
LNCFCMOCS0	B020	LNCF	2	0216
LNCFCMOCS1	B024	LNCF	2	0218
LNCFCMOCS2	B028	LNCF	2	021A
LNCFCMOCS3	B02C	LNCF	2	021C
LNCFCMOCS4	B030	LNCF	2	021E
LNCFCMOCS5	B034	LNCF	2	0220
LNCFCMOCS6	B038	LNCF	2	0222
LNCFCMOCS7	B03C	LNCF	2	0224
LNCFCMOCS8	B040	LNCF	2	0226
LNCFCMOCS9	B044	LNCF	2	0228
LNCFCMOCS10	B048	LNCF	2	022A
LNCFCMOCS11	B04C	LNCF	2	022C
LNCFCMOCS12	B050	LNCF	2	022E
LNCFCMOCS13	B054	LNCF	2	0230

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
LNCFCMOCS14	B058	LNCF	2	0232
LNCFCMOCS15	B05C	LNCF	2	0234
LNCFCMOCS16	B060	LNCF	2	0236
LNCFCMOCS17	B064	LNCF	2	0238
LNCFCMOCS18	B068	LNCF	2	023A
LNCFCMOCS19	B06C	LNCF	2	023C
LNCFCMOCS20	B070	LNCF	2	023E
LNCFCMOCS21	B074	LNCF	2	0240
LNCFCMOCS22	B078	LNCF	2	0242
LNCFCMOCS23	B07C	LNCF	2	0244
LNCFCMOCS24	B080	LNCF	2	0246
LNCFCMOCS25	B084	LNCF	2	0248
LNCFCMOCS26	B088	LNCF	2	024A
LNCFCMOCS27	B08C	LNCF	2	024C
LNCFCMOCS28	B090	LNCF	2	024E
LNCFCMOCS29	B094	LNCF	2	0250
LNCFCMOCS30	B098	LNCF	2	0252
LNCFCMOCS31	B09C	LNCF	2	0254
NOOP		LNCF	10	0256
3DSTATE_CONSTANT_VS_Commited		SVG	11	0260
3DSTATE_CONSTANT_HS_Commited		SVG	11	026B
3DSTATE_CONSTANT_DS_Commited		SVG	11	0276
3DSTATE_CONSTANT_GS_Commited		SVG	11	0281
3DSTATE_VS		SVG	9	028C
3DSTATE_BINDING_TABLE_POINTERS_VS		SVG	2	0295
3DSTATE_SAMPLER_STATE_POINTERS_VS		SVG	2	0297
3DSTATE_URB_VS		SVG	2	0299
3DSTATE_HS		SVG	9	029B
3DSTATE_BINDING_TABLE_POINTERS_HS		SVG	2	02A4
3DSTATE_SAMPLER_STATE_POINTERS_HS		SVG	2	02A6
3DSTATE_URB_HS		SVG	2	02A8
3DSTATE_TE		SVG	4	02AA
3DSTATE_DS		SVG	11	02AE
3DSTATE_BINDING_TABLE_POINTERS_DS		SVG	2	02B9

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
3DSTATE_SAMPLER_STATE_POINTERS_DS		SVG	2	02BB
3DSTATE_URB_DS		SVG	2	02BD
3DSTATE_GS		SVG	10	02BF
3DSTATE_BINDING_TABLE_POINTERS_GS		SVG	2	02C9
3DSTATE_SAMPLER_STATE_POINTERS_GS		SVG	2	02CB
3DSTATE_URB_GS		SVG	2	02CD
3DSTATE_STREAMOUT		SVG	5	02CF
3DSTATE_CLIP		SVG	4	02D4
3DSTATE_SF		SVG	4	02D8
3DSTATE_SCISSOR_STATE_POINTERS		SVG	2	02DC
3DSTATE_VIEWPORT_STATE_POINTERS_CL_SF		SVG	2	02DE
3DSTATE_RASTER		SVG	5	02E0
3DSTATE_WM_HZ_OP		SVG	5	02E5
3DSTATE_MULTISAMPLE		SVG	2	02EA
3DSTATE_CONSTANT_VS_NonComitted		SVG	11	02EC
3DSTATE_CONSTANT_HS_NonComitted		SVG	11	02F7
3DSTATE_CONSTANT_DS_NonComitted		SVG	11	0302
3DSTATE_CONSTANT_GS_NonComitted		SVG	11	030D
3DSTATE_DRAW_RECTANGULAR		SVG	4	0318
Load_Register_Immediate header	0x1100_1001	SVG	1	031C
FF_PERF_REG	0x6b1c	SVG	2	031D
NOOP		SVG	1	031F
3DSTATE_CONSTANT_PS_comitted		SVL	11	0320
NOOP		SVL	1	032B
3DSTATE_WM		SVL	2	032C
3DSTATE_VIEWPORT_STATE_POINTER_CC		SVL	2	032E
3DSTATE_CC_STATE_POINTERS		SVL	2	0330
3DSATE_WM_SAMPLEMASK		SVL	2	0332
3DSTATE_WM_DEPTH_STENCIL		SVL	4	0334
3DSTATE_WM_CHROMAKEY		SVL	2	0338
3DSTATE_DEPTH_BUFF		SVL	8	033A
3DSTATE_HIZ_DEPTH_BUFF		SVL	5	0342
3DSTATE_STC_DEPTH_BUFF		SVL	5	0347
3DSTATE_CLEAR_PARAMS		SVL	3	034C

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
3DSTATE_SBE		SVL	6	034F
3DSTATE_SBE_SWIZ		SVL	11	0355
3DSTATE_PS		SVL	12	0360
3DSTATE_BINDING_TABLE_POINTERS_PS		SVL	2	036C
STATE_SAMPLER_STATE_POINTERS_PS		SVL	2	036E
3DSTATE_BLEND_STATE_POINTERS		SVL	2	0370
3DSTATE_PS_EXTRA		SVL	2	0372
3DSTATE_PS_BLEND		SVL	2	0374
3DSTATE_CONSTANT_PS_NonComitted		SVL	11	0376
STATE_SIP		SVL	3	0381
3DSTATE_SAMPLE_PATTERN		SVL	9	0384
Load_Register_Immediate header	0x1100_101B	SVL	1	038D
Cache_Mode_0	0x7000	SVL	2	038E
Cache_Mode_1	0x7004	SVL	2	0390
GT_MODE	0x7008	SVL	2	0392
FBC_RT_BASE_ADDR_REGISTER	0x7020	SVL	2	0398
FBC_RT_BASE_ADDR_REGISTER_UPPER	0x7024	SVL	2	039A
SAMPLER_MODE	0x7028	SVL	2	039C
OA_CULL	0x7030	SVL	2	03A2
				03AA
NOOP		SVL	6	**03AA**
NOOP		TDL	1	03B0
Load_Register_Immediate header	0x1100_103B	TDL	1	03B1
TD_CTL	E400	TDL	2	03B2
TD_CTL2	E404	TDL	2	03B4
TD_VF_VS_EMSK	E408	TDL	2	03B6
TD_GS_EMSK	E40C	TDL	2	03B8
TD_WIZ_EMSK	E410	TDL	2	03BA
TD_TS_EMSK	E428	TDL	2	03BC
TD_HS_EMSK	E4B0	TDL	2	03BE
TD_DS_EMSK	E4B4	TDL	2	03C0
EU_PERF_CNT_CTL0	E458	TDL	2	03DE
EU_PERF_CNT_CTL1	E558	TDL	2	03E0
EU_PERF_CNT_CTL2	E658	TDL	2	03E2

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
EU_PERF_CNT_CTL3	E758	TDL	2	03E4
EU_PERF_CNT_CTL4	E45C	TDL	2	03E6
EU_PERF_CNT_CTL5	E55C	TDL	2	03E8
EU_PERF_CNT_CTL6	E65C	TDL	2	03EA
NOOP		TDL	2	03EE
NOOP		WM	1	03F0
Load_Register_Immediate header	0x1100_1007	WM	1	03F1
WMHWCLRVAL	0x5524	WM	2	03F6
3DSTATE_POLY_STIPPLE_PATTERN		WM	33	03FA
3DSTATE_AA_LINE_PARAMS		WM	3	041B
3DSTATE_POLY_STIPPLE_OFFSET		WM	2	041E
3DSTATE_LINE_STIPPLE		WM	3	0420
NOOP		WM	13	0423
NOOP		SC	1	0430
Load_Register_Immediate header	0x1100_1009	SC	1	0431
3DSTATE_MONOFILTER_SIZE		SC	2	**043C**
3DSTATE_CHROMA_KEY		SC	16	**043E**
NOOP		SC	2	**044E**
NOOP		DM	1	0450
3DSTATE_SAMPLER_PALETTE_LOAD0		DM	257	0451
NOOP		DM	1	0552
3DSTATE_SAMPLER_PALETTE_LOAD1		DM	257	0553
NOOP		DM	12	0654
NOOP		SOL	1	0660
Load_Register_Immediate header	0x1100_1027	SOL	1	0661
SO_NUM_PRIMS_WRITTEN0	0x5200	SOL	4	0662
SO_NUM_PRIMS_WRITTEN1	0x5208	SOL	4	0666
SO_NUM_PRIMS_WRITTEN2	0x5210	SOL	4	066A
SO_NUM_PRIMS_WRITTEN3	0x5218	SOL	4	066E
SO_PRIM_STORAGE_NEEDED0	0x5240	SOL	4	0672
SO_PRIM_STORAGE_NEEDED1	0x5248	SOL	4	0676
SO_PRIM_STORAGE_NEEDED2	0x5250	SOL	4	067A
SO_PRIM_STORAGE_NEEDED3	0x5258	SOL	4	067E
SO_WRITE_OFFSET0	0x5280	SOL	2	0682

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
SO_WRITE_OFFSET1	0x5284	SOL	2	0684
SO_WRITE_OFFSET2	0x5288	SOL	2	0686
SO_WRITE_OFFSET3	0x528C	SOL	2	0688
3DSTATE_SO_BUFFER		SOL	32	068A
NOOP		SOL	3	06AA
3DSTATE_SO_DECL_LIST		SOL	259	06AD
3DSTATE_INDEX_BUFFER		VF	5	07B0
3DSTATE_VERTEX_BUFFERS		VF	133	07B5
3DSTATE_VERTEX_ELEMENTS		VF	69	083A
3DSTATE_VF_STATISTICS		VF	1	087F
3DSTATE_VF		VF	2	0880
3DSTATE_SGVS		VF	2	0882
3DSTATE_VF_INSTANCING		VF	69	0884
3DSTATE_VF_TOPOLOGY		VF	2	08C9
			0	08CB
Load_Register_Immediate header	0x1100_1095	VF	1	08CB
INSTANCE CNT	08300 - 08384h	VF	68	08CC
INSTANCE INDX	08400 - 08484h	VF	68	0910
COMMITTED VERTEX NUMBER	08390h	VF	2	0954
COMMITTED INSTANCE ID	08394h	VF	2	0956
COMMITTED PRIMITIVE ID	08398h	VF	2	0958
STATUS	0839Ch	VF	2	095A
COMMON VERTEX	083A0h	VF	2	095C
				095E
VF_GUARDBAND	083A4h	VF	2	0960
3DSTATE_VF_COMPONENT_PACKING		VF	5	0962
				0967
NOOP		VF	9	0967
Load_Register_Immediate header	0x1100_108D	VFE	1	0970
TDL0 DATA		VFE	142	0971
NOOP		VFE	1	**09FF**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0A00**
TDL1 DATA		VFE	142	**0A01**
NOOP		VFE	1	**0A8F**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
Load_Register_Immediate header	0x1100_108D	VFE	1	**0A90**
TDL2 DATA		VFE	142	**0A91**
NOOP		VFE	1	**0B1F**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0B20**
TDL3 DATA		VFE	142	**0B21**
NOOP		VFE	1	**0BAF**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0BB0**
TDL4 DATA		VFE	142	**0BB1**
NOOP		VFE	1	**0C3F**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0C40**
TDL5 DATA		VFE	142	**0C41**
NOOP		VFE	1	**0CCF**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0CD0**
TDL6 DATA		VFE	142	**0CD1**
NOOP		VFE	1	**0D5F**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0D60**
TDL7 DATA		VFE	142	**0D61**
NOOP		VFE	1	**0DEF**
Load_Register_Immediate header	0x1100_108D	VFE	1	**0DF0**
TDL8 DATA		VFE	142	**0DF1**
NOOP		VFE	1	**0E7F**
Load_Register_Immediate header	0x1100_1085	VFE	1	**0E80**
GW0 DATA		VFE	134	**0E81**
NOOP		VFE	9	**0F07**
Load_Register_Immediate header	0x1100_1085	VFE	1	**0F10**
GW1 DATA		VFE	134	**0F11**
NOOP		VFE	9	**0F97**
Load_Register_Immediate header	0x1100_1085	VFE	1	**0FA0**
GW2 DATA		VFE	134	**0FA1**
NOOP		VFE	9	**1027**
Load_Register_Immediate header	0x1100_1085	VFE	1	**1030**
GW3 DATA		VFE	134	**1031**
NOOP		VFE	9	**10B7**
Load_Register_Immediate header	0x1100_1085	VFE	1	**10C0**

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
GW4 DATA		VFE	134	**10C1**
NOOP		VFE	9	**1147**
Load_Register_Immediate header	0x1100_1085	VFE	1	**1150**
GW5 DATA		VFE	134	**1151**
NOOP		VFE	9	**11D7**
Load_Register_Immediate header	0x1100_1085	VFE	1	**11E0**
GW6 DATA		VFE	134	**11E1**
NOOP		VFE	9	**1267**
Load_Register_Immediate header	0x1100_1085	VFE	1	**1270**
GW7 DATA		VFE	134	**1271**
NOOP		VFE	9	**12F7**
Load_Register_Immediate header	0x1100_1085	VFE	1	**1300**
GW8 DATA		VFE	134	**1301**
NOOP		VFE	9	**1387**
Load_Register_Immediate header	0x1100_1047	VFE	1	**1390**
GWC DATA		VFE	72	**1391**
NOOP		VFE	71	**13D9**
Load_Register_Immediate header	0x1100_103D	VFE	1	1420
TSG0 DATA		VFE	62	1421
NOOP		VFE	1	145F
Load_Register_Immediate header	0x1100_103D	VFE	1	1460
TSG1 DATA		VFE	62	1461
NOOP		VFE	1	149F
Load_Register_Immediate header	0x1100_103D	VFE	1	14A0
TSG2 DATA		VFE	62	14A1
NOOP		VFE	1	14DF
Load_Register_Immediate header	0x1100_103D	VFE	1	14E0
TSG3 DATA		VFE	62	14E1
NOOP		VFE	1	151F
Load_Register_Immediate header	0x1100_103D	VFE	1	1520
TSG4 DATA		VFE	62	1521
NOOP		VFE	1	155F
Load_Register_Immediate header	0x1100_103D	VFE	1	1560
TSG5 DATA		VFE	62	1561

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
NOOP		VFE	1	159F
Load_Register_Immediate header	0x1100_103D	VFE	1	15A0
TSG6 DATA		VFE	62	15A1
NOOP		VFE	1	15DF
Load_Register_Immediate header	0x1100_103D	VFE	1	15E0
TSG7 DATA		VFE	62	15E1
NOOP		VFE	1	161F
Load_Register_Immediate header	0x1100_103D	VFE	1	1620
TSG8 DATA		VFE	62	1621
NOOP		VFE	1	165F
Load_Register_Immediate header	0x1100_1001/0x1100_1003/0x1100_1009	VFE	1	1660
VFE Data		VFE	2/4/10	1661
NOOP		VFE	8/6/0	1663/1665
Load_Register_Immediate header	0x1100_1001	VFE	1	166B
GP_THREAD_TIME		VFE	2	166C
NOOP		VFE	2	166E
MEDIA_VFE_STATE		VFE	9	1670
MEDIA_POOL_STATE		VFE	6	1679
MEDIA_CURBE_LOAD		VFE	4	167F
MEDIA_INTERFACE_DESCRIPTOR_LOAD		VFE	4	1683
NOOP		VFE	9	1687
NOOP		RS	9	1690
3DSTATE_GATHER_POOL_ALLOC		RS	4	1699
3DSTATE_GATHER_CONSTANT_VS		RS	131	169D
NOOP		RS	13	1720
3DSTATE_GATHER_CONSTANT_GS		RS	131	172D
NOOP		RS	13	17B0
3DSTATE_GATHER_CONSTANT_HS		RS	131	17BD
NOOP		RS	13	1840
3DSTATE_GATHER_CONSTANT_DS		RS	131	184D
NOOP		RS	13	18D0
3DSTATE_GATHER_CONSTANT_PS		RS	131	18DD

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
NOOP		RS	2	1960
3DSTATE_BINDING_TABLE_POOL_ALLOC		RS	4	1962
3DSTATE_BINDING_TABLE_EDIT_VS		RS	258	1966
NOOP		RS	6	1A68
3DSTATE_BINDING_TABLE_EDIT_GS		RS	258	1A6E
NOOP		RS	6	1B70
3DSTATE_BINDING_TABLE_EDIT_HS		RS	258	1B76
NOOP		RS	6	1C78
3DSTATE_BINDING_TABLE_EDIT_DS		RS	258	1C7E
NOOP		RS	6	1D80
3DSTATE_BINDING_TABLE_EDIT_PS		RS	258	1D86
NOOP		RS	4	1E88
MI_BATCH_BUFFER_END/NOOP ***		RS	1	1E8C
NOOP		RS	5	1E8D
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC		RS	4	1E92
3DSTATE_DX9_CONSTANTF_VS(Global)		RS	1026	1E96
NOOP		RS	6	2298
3DSTATE_DX9_CONSTANTI_VS(Global)		RS	130	229E
NOOP		RS	6	2320
3DSTATE_DX9_CONSTANTB_VS(Global)		RS	18	2326
NOOP		RS	6	2338
3DSTATE_DX9_CONSTANTF_VS(local)		RS	1026	233E
NOOP		RS	6	2740
3DSTATE_DX9_CONSTANTI_VS(local)		RS	130	2746
NOOP		RS	6	27C8
3DSTATE_DX9_CONSTANTB_VS(local)		RS	18	27CE
NOOP		RS	3	27E0
3DSTATE_DX9_LOCAL_VALID_VS		RS	11	27E3
3DSTATE_DX9_CONSTANTF_PS(Global)		RS	1026	27EE
NOOP		RS	6	2BF0
3DSTATE_DX9_CONSTANTI_PS(Global)		RS	130	2BF6
NOOP		RS	6	2C78
3DSTATE_DX9_CONSTANTB_PS(Global)		RS	18	2C7E
NOOP		RS	6	2C90

Description	MMIO Offset/Command	Unit	# of DW	Address Offset (DWord)
3DSTATE_DX9_CONSTANTF_PS(local)		RS	1026	2C96
NOOP		RS	6	3098
3DSTATE_DX9_CONSTANTI_PS(local)		RS	130	309E
NOOP		RS	6	3120
3DSTATE_DX9_CONSTANTB_PS(local)		RS	18	3126
3DSTATE_DX9_LOCAL_VALID_PS		RS	11	3138
3DSTATE_GENERATE_ACTIVE_VS		RS	2	3143
3DSTATE_GENERATE_ACTIVE_PS		RS	2	3145
MI_BATCH_BUFFER_END		RS	1	3147
NOOP		RS	8	3148
URB_ATOMIC_STORAGE		GAFS	8192	3150
				5150
			DW	20816
			Kbytes	81.3125

Command Ordering Rules

There are several restrictions regarding the ordering of commands issued to the GPE. This subsection describes these restrictions along with some explanation of why they exist. Refer to the various command descriptions for additional information.

PIPELINE_SELECT

The previously-active pipeline needs to be flushed via the MI_FLUSH command immediately before switching to a different pipeline via use of the PIPELINE_SELECT command. Refer to Fixed and Shared Function IDs for details on the PIPELINE_SELECT command.

PIPELINE_SELECT

PIPE_CONTROL

The PIPE_CONTROL command does not require URB fencing/allocation to have been performed, nor does it rely on any other pipeline state. It is intended to be used on both the 3D pipe and the Media pipe. It has special optimizations to support the pipelining capability in the 3D pipe which do not apply to the Media pipe.

Common Pipeline State-Setting Commands

The following commands are used to set state common to both the 3D and Media pipelines. This state is comprised of CS FF unit state, non-pipelined global state (EU, etc.), and Sampler shared-function state.

- STATE_BASE_ADDRESS
- STATE_SIP
- 3DSTATE_SAMPLER_PALETTE_LOAD
- 3DSTATE_CHROMA_KEY

The state variables associated with these commands must be set appropriately prior to initiating activity within a pipeline (i.e., 3DPRIMITIVE or MEDIA_OBJECT).

3D Pipeline-Specific State-Setting Commands

The following commands are used to set state specific to the 3D Pipeline.

- 3DSTATE_PIPELINED_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- 3DSTATE_VERTEX_BUFFERS
- 3DSTATE_VERTEX_ELEMENTS
- 3DSTATE_INDEX_BUFFERS
- 3DSTATE_VF_STATISTICS
- 3DSTATE_DRAWING_RECTANGLE
- 3DSTATE_CONSTANT_COLOR
- 3DSTATE_DEPTH_BUFFER
- 3DSTATE_POLY_STIPPLE_OFFSET
- 3DSTATE_POLY_STIPPLE_PATTERN
- 3DSTATE_LINE_STIPPLE
- 3DSTATE_GLOBAL_DEPTH_OFFSET

The state variables associated with these commands must be set appropriately prior to issuing 3DPRIMITIVE.

Media Pipeline-Specific State-Setting Commands

The following command is used to set state specific to the Media pipeline:

- MEDIA_STATE_POINTERS

The state variables associated with this command must be set appropriately prior to issuing MEDIA_OBJECT.

3DPRIMITIVE

Before issuing a 3DPRIMITIVE command, all state (with the exception of MEDIA_STATE_POINTERS) needs to be valid. Thus the commands used to assigned that state must be issued before issuing 3DPRIMITIVE.

MEDIA_OBJECT

Before issuing a MEDIA_OBJECT command, all state (with the exception of 3D-pipeline-specific state) needs to be valid. Therefore the commands used to set this state need to have been issued at some point prior to the issue of MEDIA_OBJECT.

Resource Streamer

This section contains status registers and controls for the resource streamer.

Register
RS_PREEMPT_STATUS_UDW
RS_PREEMPT_STATUS - Resource Streamer Preemption Status
MI_RS_CONTEXT
MI_RS_CONTROL
MI_RS_STORE_DATA_IMM

Resource Streamer Sync Commands

Below is a table of commands that cause the resource streamer to stop and wait until the render command streamer restarts the resource streamer. If a command does not end the current batch buffer or disable the resource streamer, then the command streamer will restart the resource streamer before the next command that is used by the resource streamer.

Resource Streamer Sync Commands: Commands that RS Stops
MI_WAIT_FOR_EVENT
MI_RS_CONTROL
MI_BATCH_BUFFER_END
MI_SEMAPHORE_WAIT
MI_SET_CONTEXT
MI_RS_CONTEXT
MI_BATCH_BUFFER_START
MI_CONDITIONAL_BATCH_BUFFER_END

Introduction

The resource streamer is added to offload work from the driver without compromising on GPU optimizations. In order to reduce latency associated with these offloaded operation, H/W adds a Resource Streamer. The Resource Streamer is almost S/W invisible; S/W sees a single command stream, but it may be best for the S/W to be aware that the RS is present, as certain operations might be emphasized. The resource streamer will run ahead of the 3D Command Streamer and process only the certain commands. The Cmd steamer processes these same commands for purposes of buffer full synchronization and buffer consumption.

Glossary

No special terms identified at this time.

Common Abbreviations

Abbr.	Description/Definition
CS	**Command Streamer.** Block in charge of streaming commands. The Resource Streamer (RS) is primarily an accelerator for the CS.
FF	**Fixed Function.** Any fixed function hardward.
RS	**The Resource Streamer.** Responsible for reducing command latencies for certain command operations.
URB	**Unified Return Buffer.** The mechanism for returning information from a command.

Theory of Operation

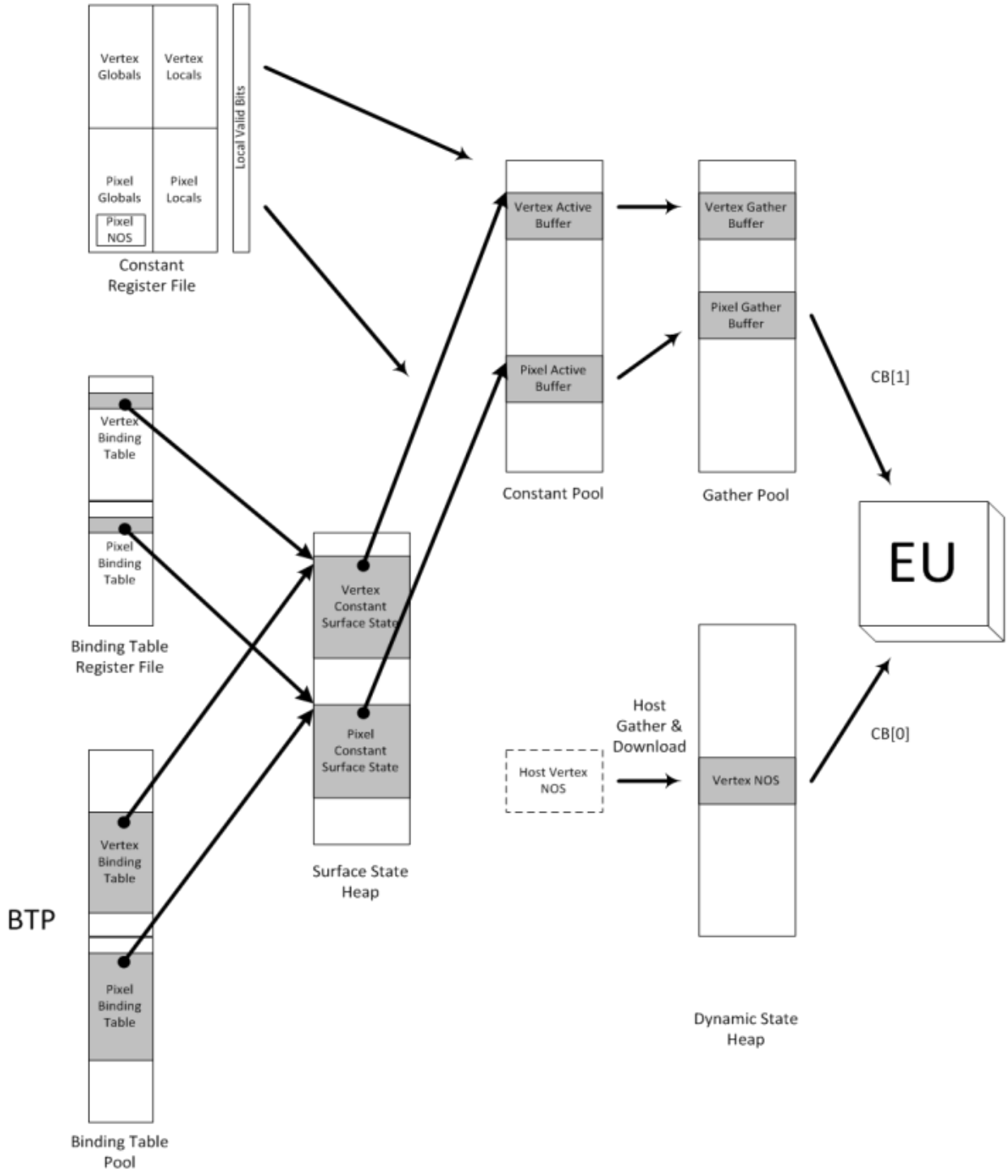
This section briefly describes the operation of the Resource Streamer. Specifically, it calls out reset state, initialization requirements, and major operational tasks of the RS.

Resource Streamer Functions

The Resource Streamer (RS) examines the commands in the ring buffer in an attempt to pre-process certain long latency items for the remainder of the graphics processing. The RS is used for the following operations:

- Batch Processing – The resource streamer reads ahead of command streamer activity to unwind batch buffers.
- Context Save – When the Command Streamer signals that context must be saved, the RS makes certain all previous cycles are completed, saves all context, and signals completion to the command streamer.
- Gather Push Constants – The RS detects GATHER commands (**3DSTATE_GATHER_POOL_ALLOC, 3DSTATE_GATHER_CONSTANT_***), and prefetchs contents needed for further command processing. The RS gets the base address of the contents by detecting the **3DSTATE_GATHER_POOL_ALLOC** command, and uses other **3DSTATE_GATHER_CONSTANT_*** commands to generate reads for data, and writes out data to memory.
- Constant Buffer Generation – Similar to other constant processes, the RS intercepts the commands for constants to update state and data.
- HW Binding Table Generation/Flush – The RS detects operations in the command stream to update binding table state and memory with bind table contents.

Resource Streamer Activity Diagram



Detailed Resource Streamer Operations

Introduction

This chapter describes the operation of the Resource Streamer in deeper detail. Most of the operations of the Resource Streamer are processed from ring buffer shown in the Ring Buffer Organization Figure in Resource Streamer Operation Descriptions. The RS examines the command stream from the ring buffer to pre-process information required by the 3D Command Streamer (CS). For a large number of the commands, the RS takes no action.

Resource Streamer Operation Descriptions

Batch Processing

When an MI_BATCH_BUFFER_START command is parsed by the render command streamer and the resource streamer enable bit is set, the command stream flags that the resource streamer is enabled. Once it parses a non-sync command then it sends the current address for where the RS must start parsing the batch buffer. The Resource Streamer parses commands until it parses a sync command, which causes the resource streamer to send a message to the command streamer that it has stopped; RS then goes idle. The command streamer stalls at a sync command until the resource streamer specifies it has stopped, and restarts the resource streamer at the next non-sync command. Below is a link to the topic with sync commands.

Commands Actions in the RS

Context Save

When the CS indicates that there is a context to be saved or restored, the RS saves its context. The CS provides an address for the RS image and issues a “batch buffer start” (see section *Batch Processing*). The RS consumes this image like any other batch buffer, and stops when it reaches the MI_BATCH_BUFFER_END command.

The context image for the Resource streamer consists of the following components:

1. HW_BINDING_TABLE_IMAGE
2. GATHER_IMAGE
3. CONSTANT_IMAGE
4. MI_BATCH_BUFFER_END

These are discussed in the following subsections.

HW Binding Table Image

While it is not always necessary to save binding table information, “split points” context switches must be saved, so the binding table contents are always saved. These consist of:

- Binding Table Generate Enable
- Binding Table Pool Base Address
- Binding Table Pool Size
- Binding Table Contents

HW Binding Table Image

Description	Dwords Required for Storage
3DSTATE_BINDING_TABLE_POOL_ALLOC	3
3DSTATE_BINDING_TABLE_EDIT_VS	194
3DSTATE_BINDING_TABLE_EDIT_GS	194
3DSTATE_BINDING_TABLE_EDIT_HS	194
3DSTATE_BINDING_TABLE_EDIT_DS	194
3DSTATE_BINDING_TABLE_EDIT_PS	194
3DSTATE_BINDING_TABLE_EDIT_VS	194

Gather Push Constants Image

Since the resource streamer does not support mid-triangle preemption, the resource steamer will have finished producing all the gather buffers by the end of the batch buffer and the cmd streamer would have consumed all the gather buffers. The following things need to be saved.

- Gather pool enable
- Gather pool base address
- Gather pool size

Therefore a 3DSTATE_GATHER_POOL_ALLOC command needs to be saved.

Gather Push Constants Image

Description	Dwords Required for Storage
3DSTATE_GATHER_POOL_ALLOC	4

Push Constant Image

We assume that the end of the batch buffer can come between any set of cmds. Therefore the following things will be saved:

- Dx9 Constant enable
- Dx9 Constant pool base address
- Dx9 Constant pool size
- Dx9 local registers (F,I,B)
- Dx9 Local Valid
- Dx9 global registers (F,I,B)

Therefore a 3DSTATE_CONSTANT_BUFFER_POOL_ALLOC command will saved. In addition, since the F register is 256 entries and only a maximum of 63 entries can be contained in a single 3DSTATE_DX9CONSTANTF_* command, 5 CONSTANTF cmds will be saved for global and 5 for local registers register per FF (VS,PS). There will be 1 3DSTATE_CONSTANTI_* will be save for global and 1 for local register per FF. There will be 1 3DSTATE_CONSTANTB_* will be save for global and 1 for local register per FF.

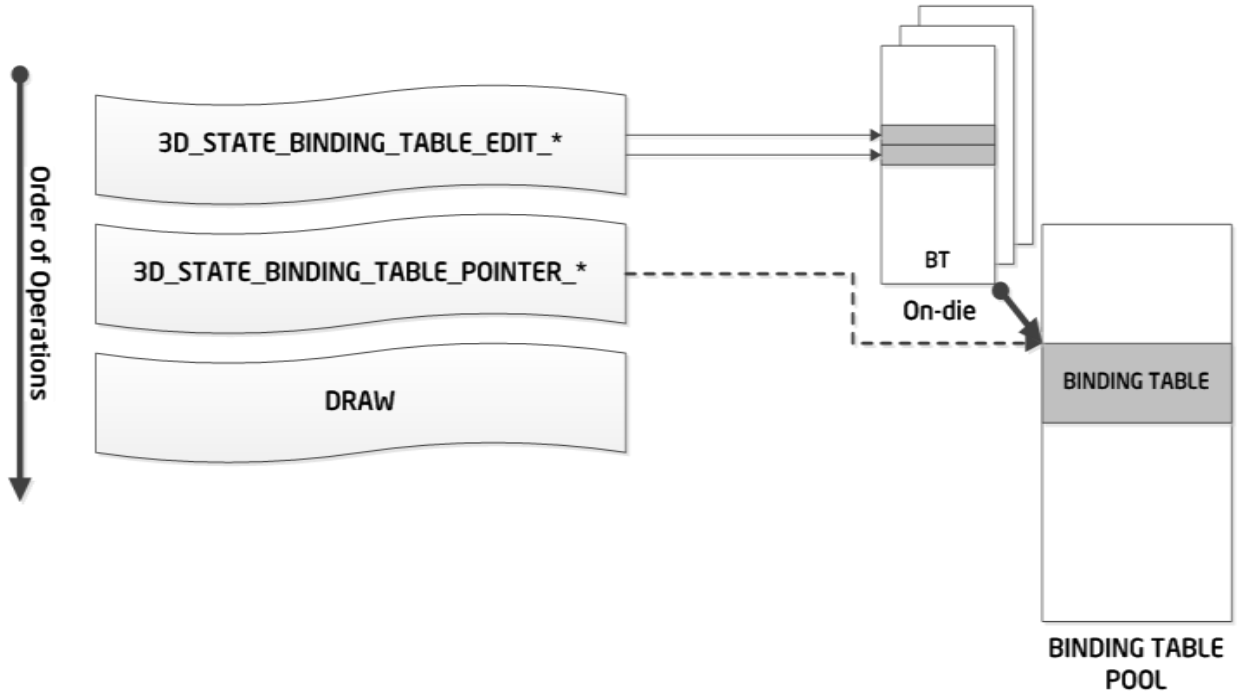
Gather Push Constants Image

Description	Dwords Required for Storage
3DSTATE_CONSTANT_BUFFER_POOL_ALLOC	4
3DSTATE_CONSTANTF_VS	1026
3DSTATE_CONSTANTI_VS	130
3DSTATE_CONSTANTB_VS	18
3DSTATE_CONSTANTF_VS	1026
3DSTATE_CONSTANTI_VS	130
3DSTATE_CONSTANTB_VS	18
3DSTATE_LOCAL_VALID_VS	10
3DSTATE_CONSTANTF_PS	1026
3DSTATE_CONSTANTI_PS	130
3DSTATE_CONSTANTB_PS	18
3DSTATE_CONSTANTF_PS	1026
3DSTATE_CONSTANTI_PS	130
3DSTATE_CONSTANTB_PS	18
3DSTATE_LOCAL_VALID_PS	10

HW Binding Table Generation

The RS generates binding tables in hardware to offload this from the driver. There is an on-die set of binding tables for each fixed-function unit (VS, GS, HS, DS, and PS). There is a set of commands generated by the driver to update each of these tables (3D_STATE_BINDING_TABLE_POINTER_*). When the RS encounters any of these commands, it writes the corresponding binding table out to the binding table pool. When the CS encounters these commands, it sends the binding table points down as pipelined state.

HW Binding Table Generation



The following table describes the different types of usages with binding table generation.

RS Active *	BT Pool Enabled	Mode
0	0	SW Generate BT in Surface State Heap
1	0	Illegal **
1	1	HW Generate BT

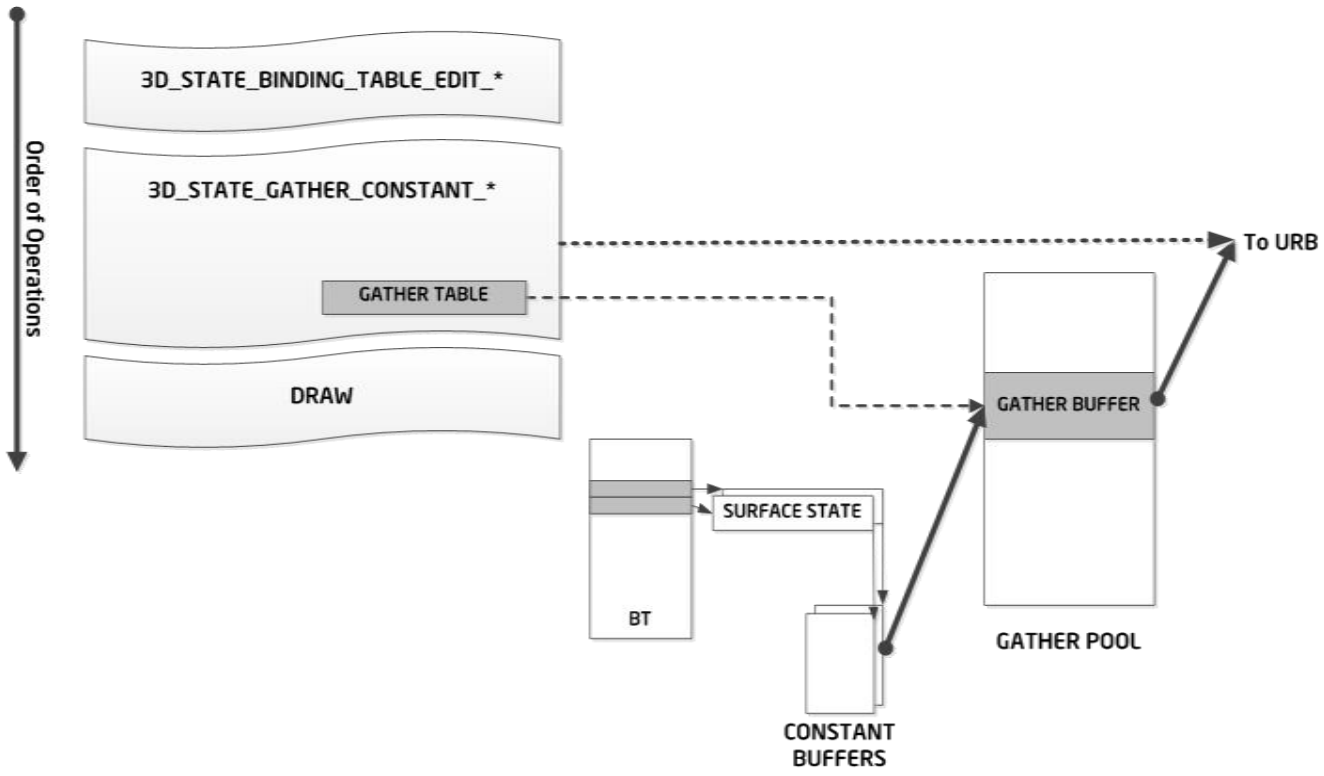
Gather Push Constants

Applications can provide up to 16 constant buffers. The compiler does some optimizations of constant usage and determines which constants should be packed in which constant register for optimal shader performance. While this gathering and packing of constant elements into push constants optimizes the shaders, it causes the driver additional work at draw call time, since the driver must gather and pack the constants at draw time.

The RS offloads the gathering process for the driver by interpreting the 3D_STATE_GATHER_CONSTANT_* for each of the fixed functions (VS, GS, DS, HS, PS). The compiler generates at gather table which instructs which elements of the buffers should be packed into the gather buffer. The gather table indexes the binding table to get a surface state which in turn points to the constant buffer. Once the gather buffer has been filled, the CS will execute the 3D_STATE_GATHER_CONSTANT_* to load the push constant into the URB.

Note: The gather push constants can ONLY BE USED if the HW generated binding tables are also used.

Gather Push Constants Generation

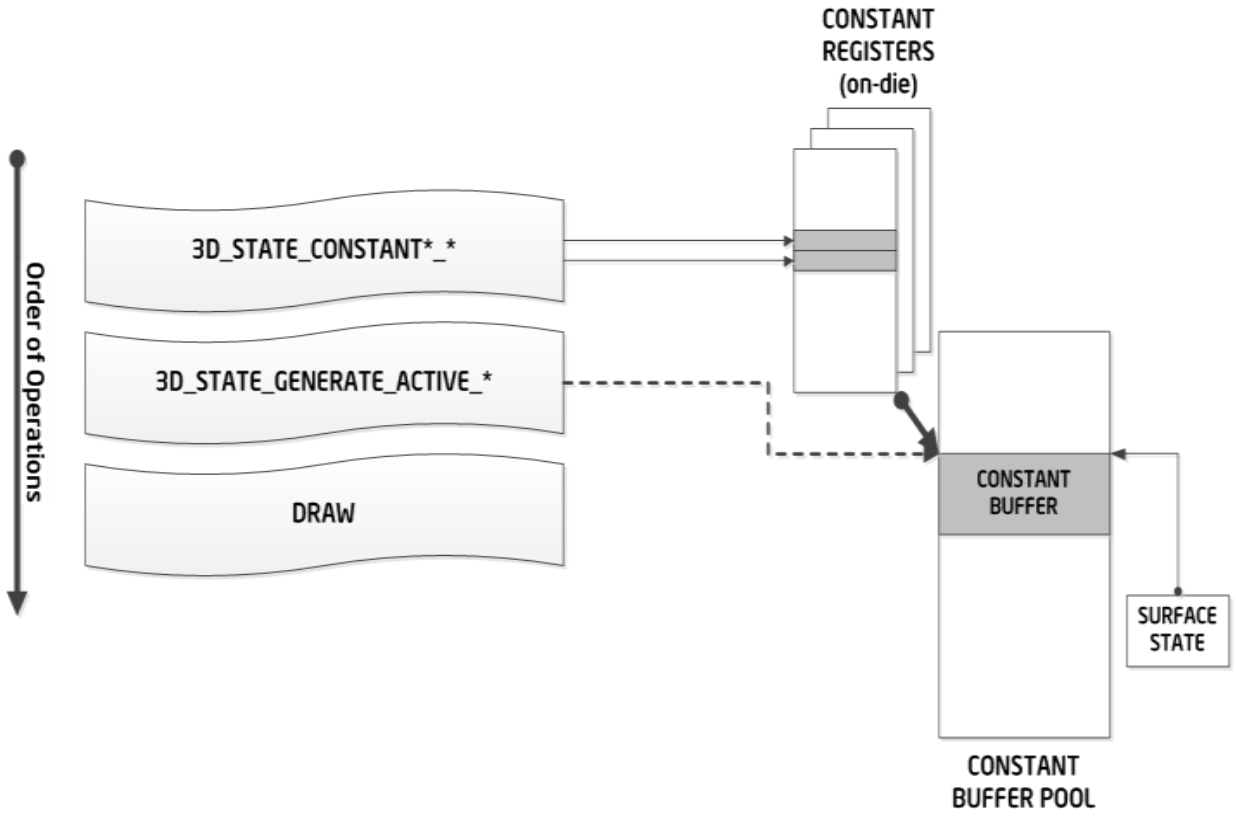


Constant Buffer Generation (not DX9)

The constant model used is a set of registers that the application can incrementally update. The hardware requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3D_STATE_CONSTANT*_* commands are used. The constant registers can be either floating, integer, or Boolean (signified by the commands CONSTANTF, CONSTANTI, or CONSTANTB, respectively). The option determines the fixed function for the constants (VS, GS, DS, HS, or PS).

When all edits to the constant registers have been completed, the 3D_STATE_GENERATE_ACTIVE_* command is used to write out a constant buffer to the Constant Buffer Pool. These buffers are fixed at 8Kbytes. Software is required to provide a surface state object that points to the constant buffer created.

Constant Buffer Generation



Commands Actions in the RS

The tables below show all 3D commands processed by the RS. In the following tables, "STOP" indicates that the RS waits for all engines to complete operations AND invalidates all command data currently in the command FIFO. "BLOCK" indicates that the RS waits for all engines to complete operation, stops further command parsing, but retains data in the command FIFO.

MI Commands Processing in the RS

Opcode (28:23)	Command	RS Handing (No Perf)	RS Handling (Perf)	Notes
03h	MI_WAIT_FOR_EVENT	STOP	BLOCK	
05h	MI_ARB_CHECK	STOP	STOP	
06h	MI_RS_CONTROL	STOP	STOP	
0Ah	MI_BATCH_BUFFER_END	STOP	STOP	
16h	MI_SEMAPHORE_MBOX	STOP	BLOCK	
18h	MI_SET_CONTEXT	STOP	STOP	
1Ah	MI_RS_CONTEXT	STOP	STOP	
31h	MI_BATCH_BUFFER_START	STOP	STOP	
36h	MI_CONDITIONAL_BATCH_BUFFER_END	STOP	STOP	

Other Commands Processed in the RS

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
0h	1h	01h	STATE_BASE_ADDRESS	RS LATCH	RS LATCH	RSunit updates the state base address if parsed
1h	1h	04h	PIPELINE_SELECT	STOP	STOP	Stop only if 3D is not selected
3h	0h	03h	Reserved			
3h	0h	04h	3DSTATE_CLEAR_PARAMS			Refer to 3D Pipeline
3h	0h	05h	3DSTATE_DEPTH_BUFFER			Refer to 3D Pipeline
3h	0h	06h	Reserved			
3h	0h	06h	3DSTATE_STENCIL_BUFFER			Refer to 3D Pipeline
3h	0h	07h	Reserved			
3h	0h	07h	3DSTATE_HIER_DEPTH_BUFFER			Refer to 3D Pipeline
3h	0h	08h	3DSTATE_VERTEX_BUFFERS			Refer to Vertex Fetch
3h	0h	09h	3DSTATE_VERTEX_ELEMENTS			Refer to Vertex Fetch
3h	0h	0Ah	3DSTATE_INDEX_BUFFER			Refer to Vertex Fetch
3h	0h	0Bh	3DSTATE_VF_STATISTICS			Refer to Vertex Fetch
3h	0h	0Ch	Reserved			
3h	0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS			Refer to 3D Pipeline
3h	0h	10h	3DSTATE_VS			Refer to Vertex Shader
3h	0h	11h	3DSTATE_GS			Refer to Geometry Shader

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	12h	3DSTATE_CLIP			Refer to Clipper
3h	0h	13h	3DSTATE_SF			Refer to Strips and Fans
3h	0h	14h	3DSTATE_WM			Refer to Windower
3h	0h	15h	3DSTATE_CONSTANT_VS			Refer to Vertex Shader
3h	0h	16h	3DSTATE_CONSTANT_GS			Refer to Geometry Shader
3h	0h	17h	3DSTATE_CONSTANT_PS			Refer to Windower
3h	0h	18h	3DSTATE_SAMPLE_MASK			Refer to Windower
3h	0h	19h	3DSTATE_CONSTANT_HS			Refer to Hull Shader
3h	0h	1Ah	3DSTATE_CONSTANT_DS			Refer to Domain Shader
3h	0h	1Bh	3DSTATE_HS			Refer to Hull Shader
3h	0h	1Ch	3DSTATE_TE			Refer to Tesselator
3h	0h	1Dh	3DSTATE_DS			Refer to Domain Shader
3h	0h	1Eh	3DSTATE_STREAMOUT			Refer to HW Streamout
3h	0h	1Fh	3DSTATE_SBE			Refer to Setup
3h	0h	20h	3DSTATE_PS			Refer to Pixel Shader
3h	0h	21h	Reserved			
3h	0h	22h	3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP			Refer to Strips & Fans

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	23h	3DSTATE_VIEWPORT_STATE_POINTERS_CC			Refer to Windower
3h	0h	24h	3DSTATE_BLEND_STATE_POINTERS			Refer to Pixel Shader
3h	0h	25h	3DSTATE_DEPTH_STENCIL_STATE_POINTERS			Refer to Pixel Shader
3h	0h	26h	3DSTATE_BINDING_TABLE_POINTERS_VS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	27h	3DSTATE_BINDING_TABLE_POINTERS_HS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	28h	3DSTATE_BINDING_TABLE_POINTERS_DS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	29h	3DSTATE_BINDING_TABLE_POINTERS_GS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	2Ah	3DSTATE_BINDING_TABLE_POINTERS_PS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	2Fh	Reserved			
3h	0h	30h	3DSTATE_URB_VS/td>	Execute	Execute	
3h	0h	31h	3DSTATE_URB_HS	Execute	Execute	
3h	0h	32h	3DSTATE_URB_DS	Execute	Execute	
3h	0h	33h	3DSTATE_URB_GS	Execute	Execute	
3h	0h	34h	3DSTATE_GATHER_VS	Execute	Execute	
3h	0h	35h	3DSTATE_GATHER_GS	Execute	Execute	
3h	0h	36h	3DSTATE_GATHER_HS	Execute	Execute	
3h	0h	37h	3DSTATE_GATHER_DS	Execute	Execute	
3h	0h	38h	3DSTATE_GATHER_PS	Execute	Execute	
3h	0h	39h	3DSTATE_CONSTANTF_VS	Execute	Execute	
3h	0h	3Ah	3DSTATE_CONSTANTF_PS	Execute	Execute	

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	3Bh	3DSTATE_CONSTANTI_VS	Execute	Execute	
3h	0h	3Ch	3DSTATE_CONSTANTI_PS	Execute	Execute	
3h	0h	3Dh	3DSTATE_CONSTANTB_VS	Execute	Execute	
3h	0h	3Eh	3DSTATE_CONSTANTB_PS	Execute	Execute	
3h	0h	3Fh	3DSTATE_LOCAL_VALID_VS	Execute	Execute	
3h	0h	40h	3DSTATE_LOCAL_VALID_PS	Execute	Execute	
3h	0h	41h	3DSTATE_GENERATE_ACTIVE_VS	Execute	Execute	
3h	0h	42h	3DSTATE_GENERATE_ACTIVE_PS	Execute	Execute	
3h	0h	43h	3DSTATE_BINDING_TABLE_EDIT_VS			Refer to Vertex Shader
3h	0h	44h	3DSTATE_BINDING_TABLE_EDIT_GS			Refer to Vertex Shader
3h	0h	45h	3DSTATE_BINDING_TABLE_EDIT_HS			Refer to Vertex Shader
3h	0h	46h	3DSTATE_BINDING_TABLE_EDIT_DS			Refer to Vertex Shader
3h	0h	47h	3DSTATE_BINDING_TABLE_EDIT_PS			Refer to Vertex Shader
3h	0h	48h	3DSTATE_VF_HASHING			
3h	0h	49h	3DSTATE_VF_INSTANCING			
3h	0h	4Ah	3DSTATE_VF_SGVS			
3h	0h	4Bh	3DSTATE_VF_TOPOLOGY			
3h	0h	4Ch	3DSTATE_WM_CHROMA_KEY			
3h	0h	4Dh	3DSTATE_PS_BLEND			
3h	0h	4Eh	3DSTATE_WM_DEPTH_STENCIL			
3h	0h	4Fh	3DSTATE_PS_EXTRA			
3h	0h	50h	3DSTATE_RASTER			
3h	0h	51h	3DSTATE_SBE_SWIZ			
3h	0h	52h	3DSTATE_WM_HZ_OP			
3h	0h	53h	3DSTATE_INT (internally generated state)			
3h	0h	54h	3DSTATE_RS_CONSTANT_POINTER			

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	55h	3DSTATE_VF_COMPONENT_PACKING			
3h	0h	57h	3DSTATE_DX9_CONSTANTF_HS			
3h	0h	58h	3DSTATE_DX9_CONSTANTI_HS			
3h	0h	59h	3DSTATE_DX9_CONSTANTB_HS			
3h	0h	5ah	3DSTATE_DX9_LOCAL_VALID_HS			
3h	0h	5bh	3DSTATE_DX9_GENERATE_ACTIVE_HS			
3h	0h	5ch	3DSTATE_DX9_CONSTANTF_DS			
3h	0h	5dh	3DSTATE_DX9_CONSTANTI_DS			
3h	0h	5eh	3DSTATE_DX9_CONSTANTB_DS			
3h	0h	5fh	3DSTATE_DX9_LOCAL_VALID_DS			
3h	0h	60h	3DSTATE_DX9_GENERATE_ACTIVE_DS			
3h	0h	61h	3DSTATE_DX9_CONSTANTF_GS			
3h	0h	62h	3DSTATE_DX9_CONSTANTI_GS			
3h	0h	63h	3DSTATE_DX9_CONSTANTB_GS			
3h	0h	64h	3DSTATE_DX9_LOCAL_VALID_GS			
3h	0h	65h	3DSTATE_DX9_GENERATE_ACTIVE_GS			
3h	0h	67h-FFh	Reserved			
3h	1h	00h	3DSTATE_DRAWING_RECTANGLE			
3h	1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0			
3h	1h	03h	Reserved			
3h	1h	04h	3DSTATE_CHROMA_KEY			
3h	1h	05h	Reserved			
3h	1h	06h	3DSTATE_POLY_STIPPLE_OFFSET			
3h	1h	07h	3DSTATE_POLY_STIPPLE_PATTERN			
3h	1h	08h	3DSTATE_LINE_STIPPLE			
3h	1h	0Ah	3DSTATE_AA_LINE_PARAMS			
3h	1h	0Bh	3DSTATE_GS_SVB_INDEX			
3h	1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1			
3h	1h	0Dh	3DSTATE_MULTISAMPLE			
3h	1h	0Eh	3DSTATE_STENCIL_BUFFER			
3h	1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER			
3h	1h	10h	3DSTATE_CLEAR_PARAMS			
3h	1h	11h	3DSTATE_MONOFILTER_SIZE			
3h	1h	12h	3DSTATE_PUSH_CONSTANT_ALLOC_VS			

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	1h	13h	3DSTATE_PUSH_CONSTANT_ALLOC_HS			
3h	1h	14h	3DSTATE_PUSH_CONSTANT_ALLOC_DS			
3h	1h	15h	3DSTATE_PUSH_CONSTANT_ALLOC_GS			
3h	1h	16h	3DSTATE_PUSH_CONSTANT_ALLOC_PS			
3h	1h	17h	3DSTATE_SO_DECL_LIST			
3h	1h	18h	3DSTATE_SO_BUFFER			
3h	1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC			
3h	1h	1Ah	3DSTATE_GATHER_POOL_ALLOC			
3h	1h	1Bh	3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC			
3h	1h	1Ch	3DSTATE_SAMPLE_PATTERN			
3h	1h	1Dh-FFh	Reserved			
3h	1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC	Execute	Execute	
3h	1h	1Ah	3DSTATE_GATHER_POOL_ALLOC	Execute	Execute	
3h	1h	1Bh	3DSTATE_CONSTANT_BUFFER_POOL_ALLOC	Execute	Execute	
3h	1h	1Ch	3DSTATE_SAMPLE_PATTERN			
3h	1h	1Dh-FFh	Reserved			
3h	2h	00h	PIPE_CONTROL			
3h	2h	01h-FFh	Reserved			
3h	3h	00h	3DPRIMITIVE	Sync	Sync	3DPRIMITIVE command is unique in that it tells the engines to send fence cycles, but does not stop RSunit (not a sync point)

Resource Streamer Programming Guidelines

This section describes RS activities and assumptions that are required for programming.

RS Interactions with the 3D Command Streamer

Because the Resource Streamer is processing ahead of the Command Streamer, many of the commands interpreted by the RS are a signal to stop further processing. In these cases, the RS completes pending activity, and waits for an indication from the Command Streamer to start again.

The specific cases that the CS commands the RS to continue are:

- Batch Buffer command parsing
- Context save

RS Interactions with Memory Requests

The RS is responsible for the generation of a number of memory requests. These are:

- Make batch buffer read requests (when address is supplied from the CS).
- Make push constant gather read requests from the state base offset.
- Make push constant gather write of packed data to the gather pool.
- Fetch the gather buffer surface base address.
- Write out the binding table pointer (BTP).
- Save BTP, constant buffer, and gather constant context data to an offset into the context image.
- Write out constant data.

As is the case in all memory accesses, read requests from the RS can be freely reordered, and may be returned in any order by the hardware. The RS consumes the cycles and presents the “software” order transparently.

When accessing the same address, a write operation followed by the read returns the written data. Writes to non-overlapping addresses may be freely reordered as well. Fencing is used to make certain all writes up to the fence have completed.

Fundamental Programming and Operational Assumptions

The following assumptions are made in the RS, and these are useful limitations to the programming:

- The CS can never send a request to a busy RS. The RS will have foreseen the situation, and stopped its operations before the CS action.
- Surface base address is never changed while in a batch buffer.
- Push constant data is 128-bit aligned.
- The GATHER command should have Constant Buffer valid bits set for any indices used in the command.

Non-Operational Activities

There are no specific events or performance counters for the resource streamer (RS).

Hardware Binding Tables

The driver spends a considerable amount of time managing the binding tables. A new command is added, 3DSTATE_BINDING_TABLE_EDIT_*, to offload the binding table generation from the driver. There is an on-die set of binding tables for each FF (VS, GS, HS, DS, PS). The 3DSTATE_BINDING_TABLE_EDIT_* commands are used by the driver to update these tables. The 3DSTATE_BINDING_TABLE_POINTER_* commands are added. When the resource streamer encounters a 3DSTATE_BINDING_TABLE_POINTER_* command, it writes the binding table out to the binding table pool. When the command streamer encounters a 3DSTATE_BINDING_TABLE_POINTER_* command, it sends the binding table pointer down as pipelined state.

Hardware Binding Tables are only supported for 3D workloads. The resource streamer must be enabled only for 3D workloads. The resource streamer must be disabled for Media and GPGPU workloads. A batch buffer containing both 3D and GPGPU workloads must take care of disabling and enabling the Resource Streamer appropriately while changing the PIPELINE_SELECT mode from 3D to GPGPU and vice versa. The resource streamer must be disabled using MI_RS_CONTROL command and Hardware Binding Tables must be disabled by programming 3DSTATE_BINDING_TABLE_POOL_ALLOC with "Binding Table Pool Enable" set to disable (i.e value '0'). The following example shows disabling and enabling of the resource streamer in a batch buffer for 3D and GPGPU workloads:

```
MI_BATCH_BUFFER_START (Resource Streamer Enabled)
PIPELINE_SELECT (3D)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled)
3D WORKLOAD MI_RS_CONTROL (Disable Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Disabled)
PIPELINE_SELECT (GPGPU)
GPGPU Workload
PIPELINE_SELECT (3D)
MI_RS_CONTROL (Enable Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled)
3D WORKLOAD
MI_BATCH_BUFFER_END
```

3DSTATE_BINDING_TABLE_POOL_ALLOC

Programming Note
The binding table generator feature has a simple all or nothing model. If HW generated binding tables are enabled, the driver must enable the pool and use 3D_HW_BINDING_TABLE_POINTER_* commands.
When switching between HW and SW binding table generation, SW must issue a state cache invalidate.
A maximum of 16,383 Binding Tables are allowed in any batch buffer.

The variable length commands are 3DSTATE_BINDING_TABLE_EDIT_HS, 3DSTATE_BINDING_TABLE_EDIT_DS, and 3DSTATE_BINDING_TABLE_EDIT_PS.

3DSTATE_BINDING_TABLE_POOL_ALLOC

3DSTATE_BINDING_TABLE_EDIT_VS

3DSTATE_BINDING_TABLE_EDIT_HS

3DSTATE_BINDING_TABLE_EDIT_DS

3DSTATE_BINDING_TABLE_EDIT_GS

3DSTATE_BINDING_TABLE_EDIT_PS

Gather Constants

Gather commands support fetching from 16 different constant buffers or one constant buffer of 8KB size. The compiler does some optimizations of constant usage and determines which elements of which constants should be packed in which push constant register for optimum shader performance. While this gathering and packing of constant elements into push constant registers optimizes the shader, it causes the driver added work at draw call time, because the driver must do the gather and packing at draw time. New commands(3D_STATE_GATHER_CONSTANT_* and 3DSTATE_GATHER_POOL_ALLOC) were added to offload the gather and packing functions from the driver. The base address for the push constant buffer and the enabling of the feature is programmed through the 3DSTATE_GATHER_POOL_ALLOC. There are 5 FF which support push constants (VS, GS, DS, HS, PS) and they all have corresponding gather commands. The compiler generates a gather table that specifies what elements of what buffers are packed into the gather buffer. The gather table indexes the BT to get the surface state which points to the constant buffer. The resource streamer gathers constants by reading the constant buffer, packs the data and then writes the buffer out to a push constant buffer based on the base address and the offset in the 3DSTATE_GATHER_CONSTANT_* command.

3DSTATE_GATHER_POOL_ALLOC

3DSTATE_GATHER_CONSTANT_VS

3DSTATE_GATHER_CONSTANT_HS

3DSTATE_GATHER_CONSTANT_DS

3DSTATE_GATHER_CONSTANT_GS

3DSTATE_GATHER_CONSTANT_PS

Workarounds

Workaround
Commands executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer valid greater than zero.

Dx9 Constant Buffer Generation

The Dx9 constant model is a set of register that the App can incrementally update. The HW requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3DSTATE_DX9_CONSTANT*_* cmds are added. These commands allow the on-die constant register to be maintained. When all the edits to the constant register have been completed, the 3DSTATE_DX9_GENERATE_ACTIVE_* cmd is used to write out a constant buffer to the Dx9 Constant buffer pool. The Dx9 constant buffers are fixed 8KB in size, with a large portion of the second 4KB unused.

Programming Note	
Context:	Dx9 Constant Buffer generation
For buffers, which have no inherent "height," padding requirements are different. A buffer must be padded to the next multiple of 256 array elements, with an additional 16 bytes added beyond that to account for the L1 cache line.	

3DSTATE_RS_CONSTANT_POINTER

Programming Note	
Context:	Dx9 Constant Buffer generation.
• The Dx9 constant buffer feature has a simple all or nothing model. • A maximum of 16,383 Binding Tables are allowed in any batch buffer. • The Dx9 constants can only be enabled if the binding table generator is also enabled.	

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

Vertex Shader Constant

This section contains various commands for the vertex shader constant.

3DSTATE_DX9_CONSTANTF_VS

3DSTATE_DX9_CONSTANTI_VS

3DSTATE_DX9_CONSTANTB_VS

3DSTATE_DX9_LOCAL_VALID_VS

3DSTATE_DX9_GENERATE_ACTIVE_VS

Pixel Shader Constant

This section contains various commands for the pixel shader constant.

3DSTATE_DX9_CONSTANTF_PS

3DSTATE_DX9_CONSTANTI_PS

3DSTATE_DX9_CONSTANTB_PS

3DSTATE_DX9_LOCAL_VALID_PS

3DSTATE_DX9_GENERATE_ACTIVE_PS

Shared Functions

Shared Functions are hardware units that perform operations on behalf a thread running in an EU. Shared Functions are sent a message payload by the thread, and optionally return results to the thread.

The most common shared function operations are memory surface read and write messages performed by the Sampler, the Pixel Port, and the Data Port. The following sections describe each of the shared function operations in detail.

Shared functions are identified by their Shared Function ID (SFID), which is one of the parameters to the EU send instruction.

List of Shared Function Identifiers

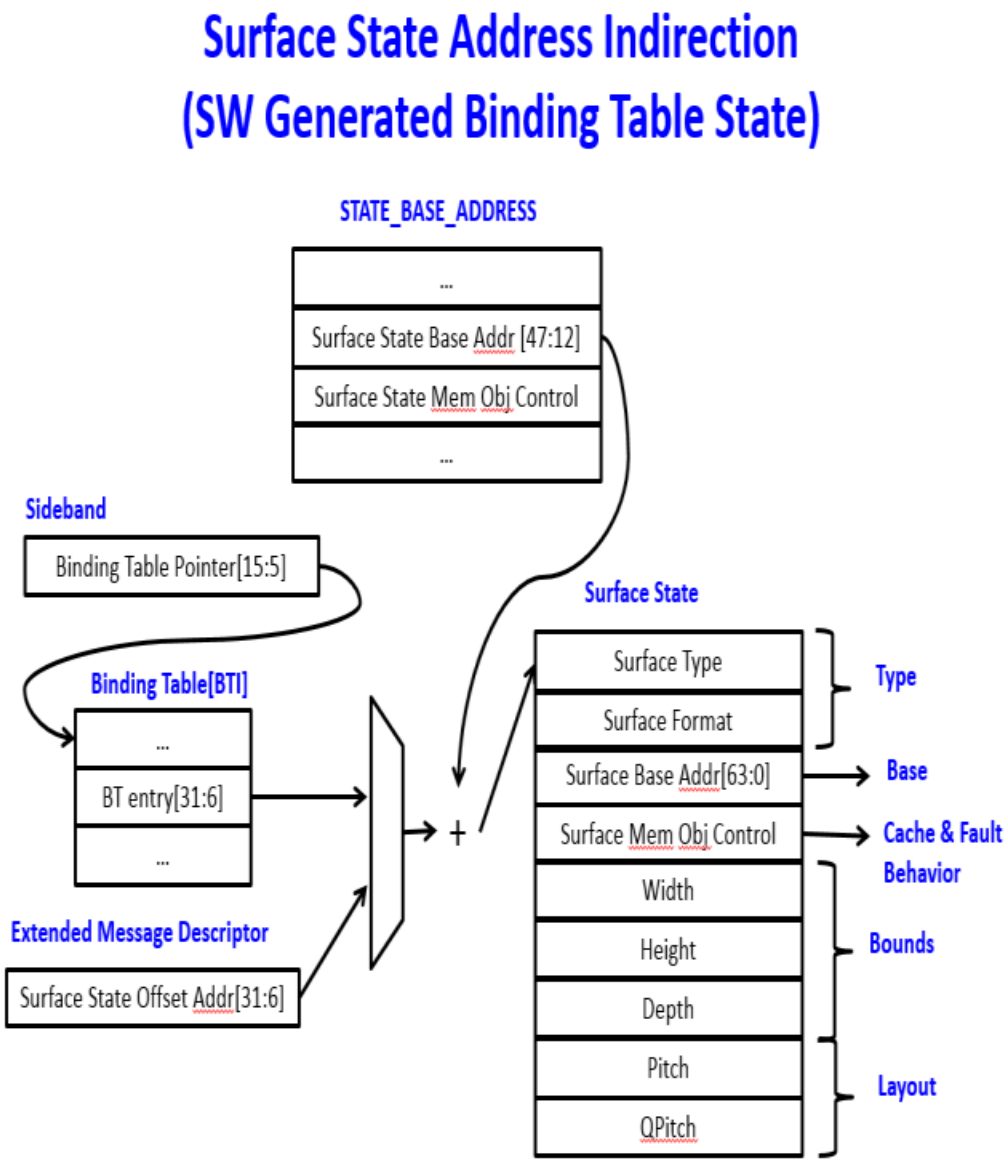
Programming Note	
Context:	Shared Function ID
SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more messages types. They act as a single logical entity.	

Programming Note	
Context:	Shared Function ID
SFID_DP_DC1 , SFID_DP_DC2, and SFID_P_DC3 are extensions of SFID_DP_DC0 to allow for more messages types. They act as a single logical entity.	

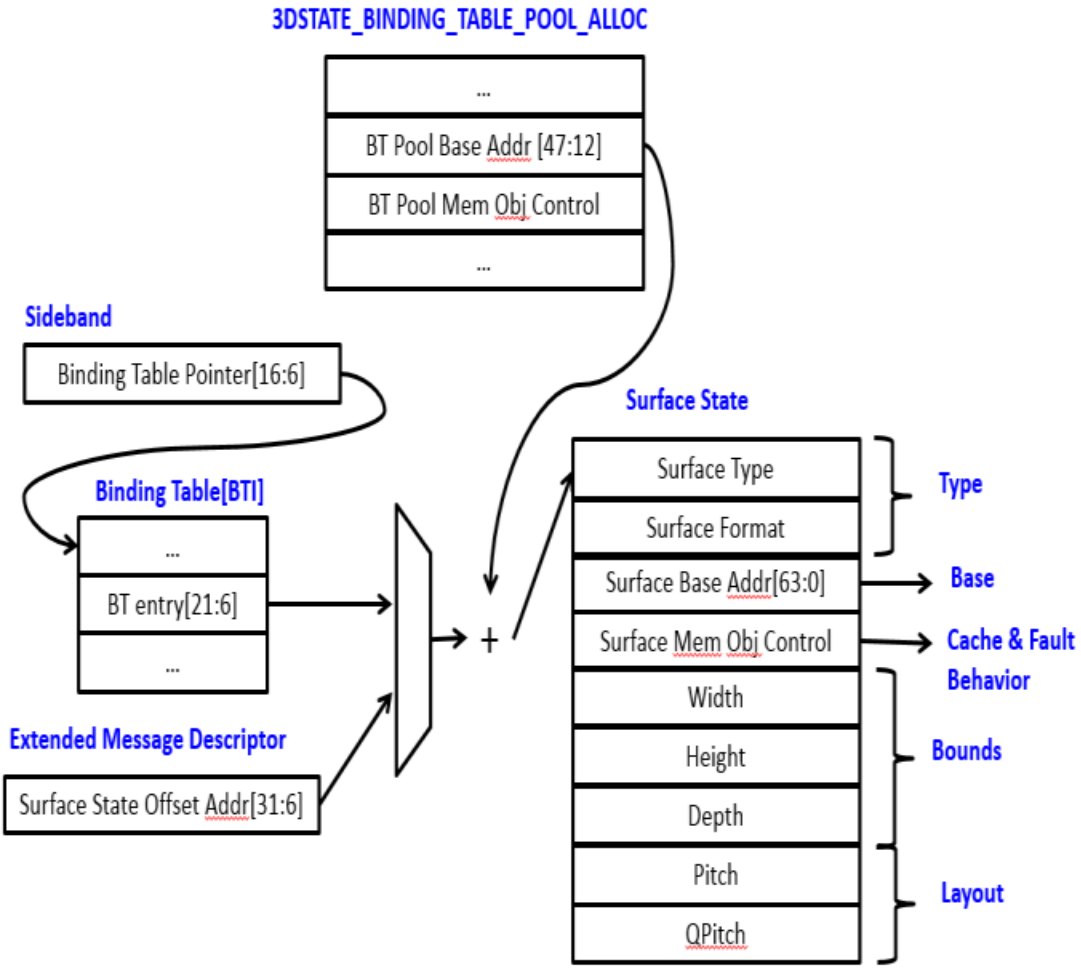
Binding Table

The surface state model is intended to be used for constant buffers, render target surfaces, and media surfaces.

Surface State Model Memory Characteristics



Surface State Address Indirection (HW Generated Binding Table)



Below is a table specifying the different configuratiosn of the Binding Table Pointers and the BINDING_TABLE_STATE.

Binding Table Pool Enable	Binding Table Entry Format	Binding Table Pointer	Base Address
0	32 bit size/[31:6]	15:5	Surface State Base Address
1	16 bit size/[21:6]	16:6	Binding Table Pool Address

Filtered Table (put table title on this line)

The data port uses the binding table to retrieve surface state when the Binding Table Index is less than 240. See the following for a definition of the binding table:

- **SW Generated BINDING_TABLE_STATE**
- **HW Generated BINDING_TABLE_STATE**

3D Sampler

The 3D Sampling Engine provides the capability of advanced sampling and filtering of surfaces in memory.

The sampling engine function is responsible for providing filtered texture values to the Gen Core in response to sampling engine messages. The sampling engine uses SAMPLER_STATE to control filtering modes, address control modes, and other features of the sampling engine. A pointer to the sampler state is delivered with each message, and an index selects one of 16 states pointed to by the pointer. Some messages do not require SAMPLER_STATE. In addition, the sampling engine uses SURFACE_STATE to define the attributes of the surface being sampled. This includes the location, size, and format of the surface as well as other attributes.

Although data is commonly used for “texturing” of 3D surfaces, the data can be used for any purpose once returned to the execution core. The 3D Sampler can be used to assist the media sampler in specific operations such as video scaling.

The following table summarizes the various subfunctions provided by the Sampling Engine. After the appropriate subfunctions are complete, the 4-component (reduced to fewer components in some cases) filtered texture value is provided to the Gen Core to complete the *sample* instruction.

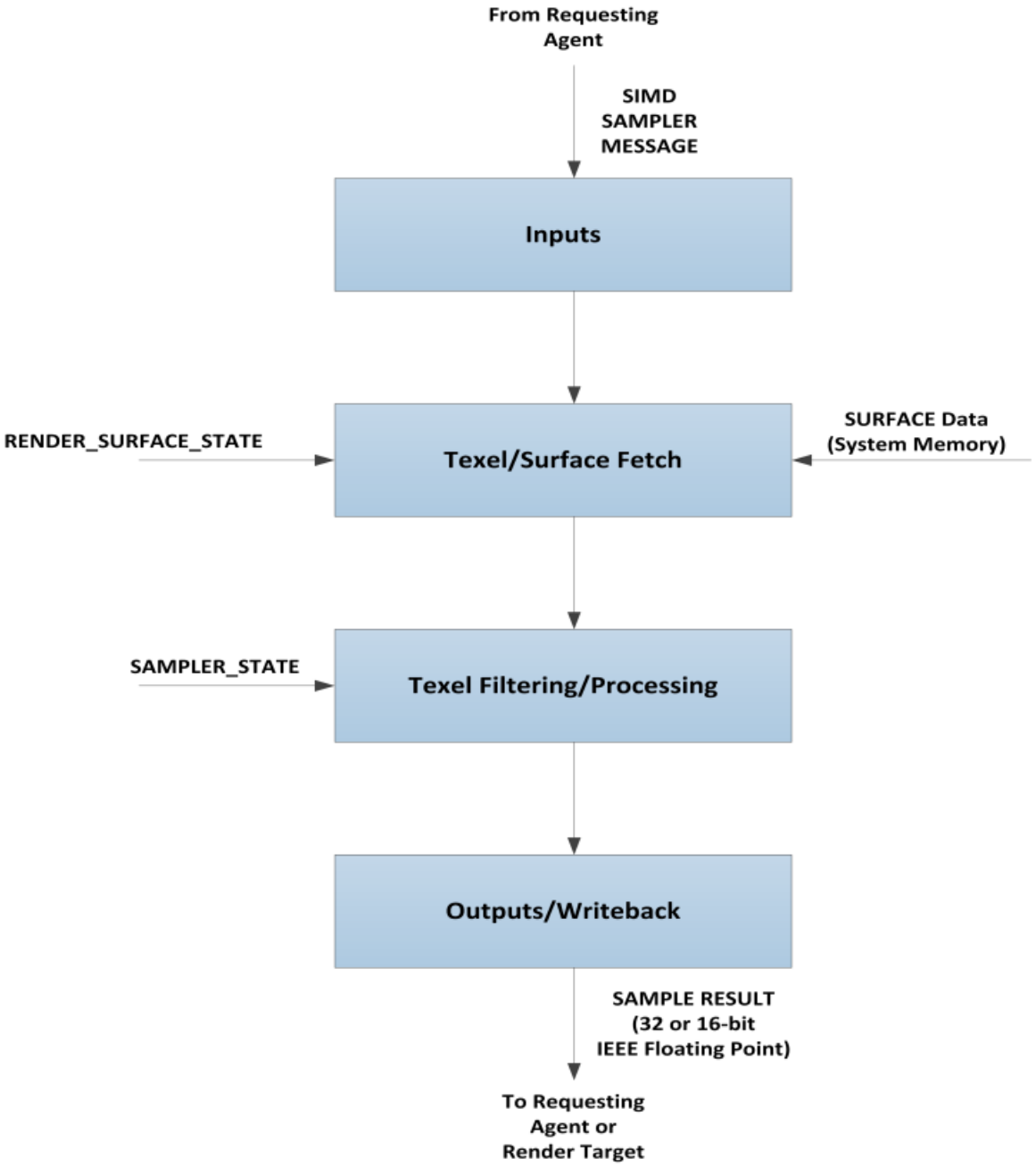
Subfunction	Description
Texture Coordinate Processing	Any required operations are performed on the incoming pixel's interpolated internal texture coordinates. These operations may include cube map intersection.
Texel Address	The Sampling Engine determines the required set of texel samples (specific texel values from

Subfunction	Description
Generation	specific texture maps), as defined by the texture map parameters and filtering modes. This includes coordinate wrap/clamp/mirror control, mipmap LOD computation and sample and/or miplevel weighting factors to be used in the subsequent filtering operations.
Texel Fetch	The required texel samples are read from the texture map. This step may require decompression of texel data. The texel sample data is converted to an internal format.
Texture Palette Lookup	For streams which have "paletted" texture surface formats, this function uses the "index" values read from the texture map to look up texel color data from the texture palette.
Shadow Pre-Filter Compare	For shadow mapping, the texel samples are first compared to the 3rd (R) component of the pixel's texture coordinate. The boolean results are used in the texture filter.
Texel Filtering	Texel samples are combined using the filter weight coefficients computed in the Texture Address Generation function. This "combination" ranges from simply passing through a "nearest" sample to blending the results of anisotropic filters performed on two mipmap levels. The output of this function is a single 4-component texel value.
Texel Color Gamma Linearization	Performs optional gamma decorrection on texel RGB (not A) values.
8x8 Video Scaler	Performs scaling using an 8x8 filter.

Sampling Engine

3D Sampler Theory of Operation

The 3D sampler (sometimes referred to as texture sampler) is a self-contained functional block within the Graphics Core which receives messages from other agents in the Graphics Core, fetches data from external memory sources typically referred to as "surfaces", performs operations on the data and returns the results in standard formats to the requester (or directly to a Render Target is requested). One of the most common applications of the 3D sampler is to return a filtered/blended pixel from a location in a texture map.



Sampler Inputs Messages

Input requests to the 3D Sampler are in the form of messages (see Messages sub-section for a description of message types and formats). A pixel shader kernal executing on the Graphics Core is an example of an agent which is capable of sending sample messages to the 3D Sampler.

In its most basic form, the sampler receives coordinates to a location within a field of data (often a texture map or depth map) and returns a value which represents the pixel color or depth which may be

filtered/blended as defined by associated surface and sampler state objects. Sampler can also work on un-typed data structures called buffers.

Messages are sent in SIMD (Single Instruction Multiple Data) format where there are 8, 16, 32 or 64 coordinate tuples to be processed (i.e. SIMD8, SIMD16 etc.) in the same manner. Some message types are restricted to SIMD8 and SIMD16 varieties and other are restricted to SIMD32 or SIMD64. See the section on Texture Coordinate Processing more details on texture coordinate requirements.

SIMD8 and SIMD16 messages are further organized into groups of 4 sets of coordinates which generally form a 2x2 “subspan” of texel locations. The spatial locality of the texel locations within a sub-span improves the performance of the sampler and allows the processing of the 4 texel locations in parallel. A SIMD8 message contains two subspans and a SIMD16 contains 4 subspans.

Sampler Data Fetches

The 3D sampler will automatically fetch required data from surfaces in system memory as needed to perform each sample operation. Fetched data may be stored in an internal cache to reduce latency for subsequent fetch operations.

The sampler calculates the address into a surface and uses RENDER_SURFACE_STATE state objects to determine the location within system memory and the format of the surface being fetched. Sampler can also receive or calculate the LOD (Level of Detail) of a surface if the surface supports multiple Mips and will fetch from the correct Mip in this case. See Texture Address Calculation sub-section for more detail on addresses and LOD calculation.

The sampler will also automatically decompress any supported compression format once data has been fetched. See the subsection Surface State for a list of supported surface formats, including compressed formats. Likewise, the sampler can linearize (inverse Gamma) sRGB formats prior to filtering.

Sampler Filtering and Processing

The sampler is capable of performing all basic filtering operations (point, bilinear, trilinear, anisotropic, cube etc.) based on the SAMPLER_STATE state object associated with the sample operation being requested.

In most cases, data returned is in the form of 32-bit or 16-bit IEEE floating-point per channel to ensure maximum precision. See Writeback Message section for a description of the format of returned data. Output Data is only returned to the requesting agent or written to a designated Render Target (RT). Sample results are never cached within the sampler or written to system memory.

Texture Coordinate Processing

The Texture Coordinate Processing function of the Sampling Engine performs any operations on the texture coordinates that are required before physical addresses of texel samples can be generated.

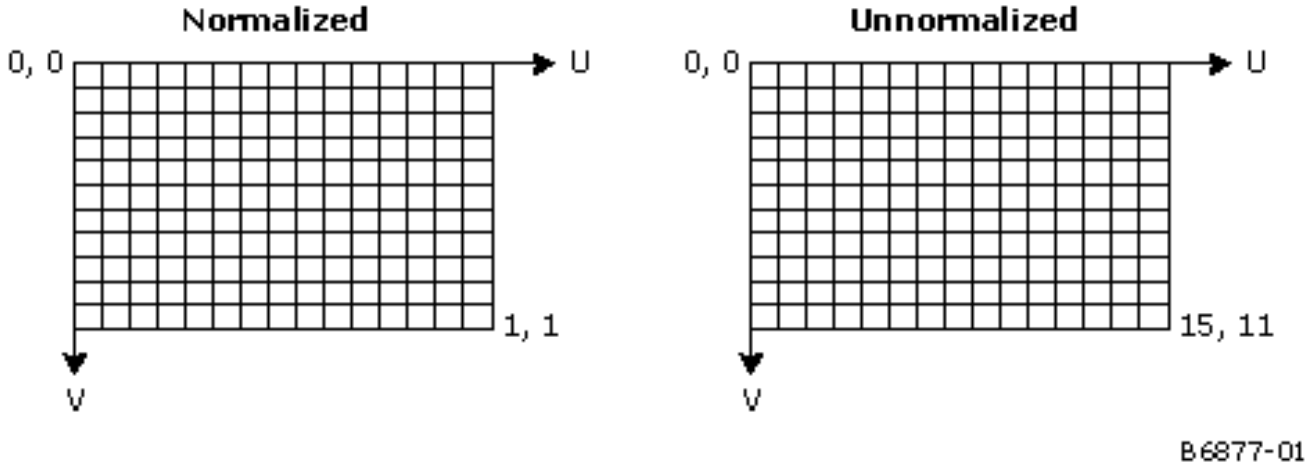
Texture Coordinate Normalization

A texture coordinate may have *normalized* or *unnormalized* values. In this function, unnormalized coordinates are normalized.

Normalized coordinates are specified in units relative to the map dimensions, where the origin is located at the upper/left edge of the upper left texel, and the value 1.0 coincides with the lower/right edge of the lower right texel . 3D rendering typically utilizes normalized coordinates.

Unnormalized coordinates are in units of texels and have not been divided (normalized) by the associated map's height or width. Here the origin is the located at the upper/left edge of the upper left texel of the base texture map.

Normalized vs. Unnormalized Texture Coordinates



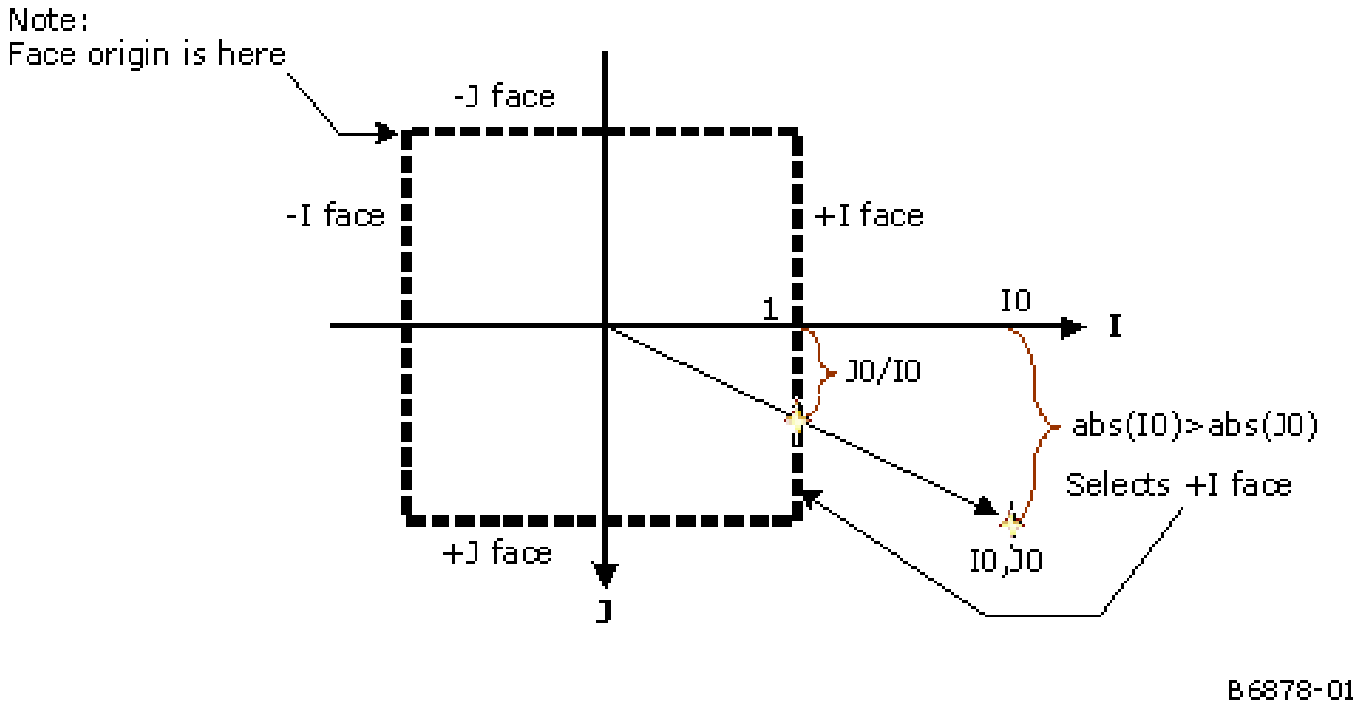
Texture Coordinate Computation

Cartesian (2D) and homogeneous (projected) texture coordinate values are projected from (interpolated) screen space back into texture coordinate space by dividing the pixel's S and T components by the Q component. This operation is done prior to sending sample operations to the 3D sampler.

Vector (cube map) texture coordinates are generated by first determining which of the 6 cube map faces (+X, +Y, +Z, -X, -Y, -Z) the vector intersects. The vector component (X, Y or Z) with the largest absolute value determines the proper (major) axis, and then the sign of that component is used to select between the two faces associated with that axis. The coordinates along the two minor axes are then divided by the coordinate of the major axis, and scaled and translated, to obtain the 2D texture coordinate ([0,1]) within the chosen face. Note that the coordinates delivered to the sampling engine must already have been divided by the component with the largest absolute value.

An illustration of this cube map coordinate computation, simplified to only two dimensions, is provided below:

Cube Map Coordinate Computation Example



Sampler SW performance hints

This is a section describing things that SW can do to get better performance out of sampler hardware.

I,L,LA,A formats

These are expanded to RGBA formats in sampler and as result are treated as larger texel sizes in sampler and have worse performance. Most of the time it is better to use the Shader Channel Select signals in RENDER_SURFACE_STATE to emulate them using R or RG formats.

		Channel select			
Base format	New format	R	G	B	A
I*	R*	R	R	R	R
L*	R*	R	R	R	1.0
LA*	RG*	R	R	R	G
A*	R*	0.0	0.0	0.0	R

Issues with this where result can be different. The A channel for LA and A format will be 0.0 rather than 1.0.

The table below shows the mapping requires for all Luminance, Intensity and Transparency formats:

Original Format	Mapped To Format	Channel select Programming (R/G/B/A)
A16_FLOAT	R16_FLOAT	0/0/0/R
A16_UNORM	R16_UNORM	0/0/0/R
A24X8_UNORM	R24_UNORM_X8_TYPELESS	0/0/0/R
A32_FLOAT	R32_FLOAT	0/0/0/R
A32_UNORM	R32_UNORM	0/0/0/R
A8_UNORM	R8_UNORM	0/0/0/R
I16_FLOAT	R16_FLOAT	R/R/R/R
I16_UNORM	R16_UNORM	R/R/R/R
I24X8_UNORM	R24_UNORM_X8_TYPELESS	R/R/R/R
I32_FLOAT	R32_FLOAT	R/R/R/R
I8_SINT	R8_SINT	R/R/R/R
I8_UINT	R8_UINT	R/R/R/R
I8_UNORM	R8_UNORM	R/R/R/R
L16_FLOAT	R16_FLOAT	R/R/R/1
L16_UNORM	R16_UNORM	R/R/R/1
L16A16_FLOAT	R16G16_FLOAT	R/R/R/G
L16A16_UNORM	R16G16_UNORM	R/R/R/G
L24X8_UNORM	R24_UNORM_X8_TYPELESS	R/R/R/1
L32_FLOAT	R32_FLOAT	R/R/R/1
L32_UNORM	R32_UNORM	R/R/R/1
L32A32_FLOAT	R32G32_FLOAT	R/R/R/G
L8_SINT	R8_SINT	R/R/R/1
L8_UINT	R8_UINT	R/R/R/1
L8_UNORM	R8_UNORM	R/R/R/1
L8A8_SINT	R8G8_SINT	R/R/R/G
L8A8_UINT	R8G8_UINT	R/R/R/G
L8A8_UNORM	R8G8_UNORM	R/R/R/G
Y8_UNORM	R8_UNORM	R/R/R/1

Tile Mode

If in TILEMODE_LINEAR and the base_address is NOT cacheline aligned there will be a very large performance hit in addition to the one that is already on linear.

Write channel mask

Starting in SKL a header is no longer needed to specify a channel mask other than all enabled. If head is disabled the response length is used to determine a default channel mask. See message format section for more details.

LD* Messages

NativeL1 Fill rate improvement SW requirements

Texel Address Generation

To better understand texture mapping, consider the mapping of each object (screen-space) pixel onto the textures images. In texture space, the pixel becomes some arbitrarily sized and aligned quadrilateral. Any given pixel of the object may “cover” multiple texels of the map, or only a fraction of one texel. For each pixel, the usual goal is to sample and filter the texture image in order to best represent the covered texel values, with a minimum of blurring or aliasing artifacts. Per-texture state variables are provided to allow the user to employ quality/performance/footprint tradeoffs in selecting how the particular texture is to be sampled.

The Texel Address Generation function of the Sampling Engine is responsible for determining how the texture maps are to be sampled. Outputs of this function include the number of texel to be fetched, along with the physical addresses of the samples and the filter weights to be applied to the samples after they are read. This information is computed given the incoming texture coordinate and gradient values, and the relevant state variables associated with the sampler and surface. This function also applies the texture coordinate address controls when converting the sample texture coordinates to map addresses.

Level of Detail Computation (Mipmapping)

Due to the specification and processing of texture coordinates at object vertices, and the subsequent object warping due to a perspective projection, the texture image may become *magnified* (where a texel covers more than one pixel) or *minified* (a pixel covers more than one texel) as it is mapped to an object. In the case where an object pixel is found to cover multiple texels (texture minification), merely choosing one (e.g., the texel sample nearest to the pixel’s texture coordinate) will likely result in severe aliasing artifacts.

Mipmapping and texture filtering are techniques employed to minimize the effect of undersampling these textures. With mipmapping, software provides *mipmap levels,* a series of pre-filtered texture maps of decreasing resolutions that are stored in a fixed (monolithic) format in memory. When mipmaps are provided and enabled, and an object pixel is found to cover multiple texels (e.g., when a textured object is located a significant distance from the viewer), the device will sample the mipmap level(s) offering a texel/pixel ratio as close to 1.0 as possible.

The device supports up to 14 mipmap levels per map surface, ranging from 8192 x 8192 texels to a 1 X 1 texel. Each successive level has ½ the resolution of the previous level in the U and V directions (to a minimum of 1 texel in either direction) until a 1x1 texture map is reached. The dimensions of mipmap levels need not be a power of 2.

Each mipmap level is associated with a *Level of Detail (LOD)* number. LOD is computed as the approximate, $\log_2$ measure of the ratio of texels per pixel. The highest resolution map is considered LOD 0. A larger LOD number corresponds to lower resolution mip level.

The *Sampler[]BaseMipLevel* state variable specifies the LOD value at which the minification filter vs. the magnification filter should be applied.

When the texture map is magnified (a texel covers more than one pixel), the base map (LOD 0) texture map is accessed, and the magnification mode selects between the nearest neighbor texel or bilinear interpolation of the 4 neighboring texels on the base (LOD 0) mipmap.

Base Level Of Detail (LOD)

The per-pixel LOD is computed in an implementation-dependent manner and approximates the $\log_2$ of the texel/pixel ratio at the given pixel. The computation is typically based on the differential texel-space distances associated with a one-pixel differential distance along the screen x- and y-axes. These texel-space distances are computed by evaluating neighboring pixel texture coordinates, these coordinates being in units of texels on the base MIP level (multiplied by the corresponding surface size in texels). The q coordinates represent the third dimension for 3D (volume) surfaces, this coordinate is a constant 0 for 2D surfaces.

The ideal LOD computation is included below.

$$LOD(x,y) = \log_2[\rho(x,y)]$$

where :

$$\rho(x,y) = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial q}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial q}{\partial y}\right)^2}\right\},$$

LOD Bias

A biasing offset can be applied to the computed LOD and used to artificially select a higher or lower miplevel and/or affect the weighting of the selected mipmap levels. Selecting a slightly higher mipmap level will trade off image blurring with possibly increased performance (due to better texture cache reuse). Lowering the LOD tends to sharpen the image, though at the expense of more texture aliasing artifacts.

The LOD bias is defined as sum of the *LODBias* state variable and the *pixLODBias* input from the input message (which can be non-zero only for sample_b messages). The application of LOD Bias is

unconditional, therefore these variables must both be set to zero in order to prevent any undesired biasing.

Note that, while the LOD Bias is applied prior to clamping and min/mag determination and therefore can be used to control the min-vs-mag crossover point, its use has the undesired effect of actually changing the LOD used in texture filtering.

LOD Pre-Clamping

The LOD Pre-Clamping function can be enabled or disabled via the *LODPreClampEnable* state variable. Enabling pre-clamping matches OpenGL semantics .

After biasing and/or adjusting of the LOD , the computed LOD value is clamped to a range specified by the (integer and fractional bits of) *MinLOD* and *MaxLOD* state variables prior to use in Min/Mag Determination.

MaxLOD specifies the lowest resolution mip level (maximum LOD value) that can be accessed, even when lower resolution maps may be available. Note that this is the only parameter used to specify the number of valid mip levels that be can be accessed, i.e., there is no explicit "number of levels stored in memory" parameter associated with a mip-mapped texture. All mip levels from the base mip level map through the level specified by the integer bits of *MaxLOD* must be stored in memory, or operation is UNDEFINED.

MinLOD specifies the highest resolution mip level (minimum LOD value) that can be accessed, where LOD==0 corresponds to the base map. This value is primarily used to deny access to high-resolution mip levels that have been evicted from memory when memory availability is low.

MinLOD and *MaxLOD* have both integer and fractional bits. The fractional parts will limit the inter-level filter weighting of the highest or lowest (respectively) resolution map. For example if *MinLOD* is 4.5 and *MipFilter* is LINEAR, LOD 4 can contribute only up to 50% of the final texel color.

Min/Mag Determination

The biased and clamped LOD is used to determine whether the texture is being minified (scaled down) or magnified (scaled up).

The *BaseMipLevel* state variable is subtracted from the biased and clamped LOD. The *BaseMipLevel* state variable therefore has the effect of selecting the "base" mip level used to compute Min/Mag Determination. (This was added to match OpenGL semantics). Setting *BaseMipLevel* to 0 has the effect of using the highest-resolution mip level as the base map.

If the biased and clamped LOD is non-positive, the texture is being magnified, and a single (high-resolution) miplevel will be sampled and filtered using the *MagFilter* state variable. At this point the computed LOD is reset to 0.0. Note that LOD Clamping can restrict access to high-resolution miplevels.

If the biased LOD is positive, the texture is being minified. In this case the *MipFilter* state variable specifies whether one or two mip levels are to be included in the texture filtering, and how that (or those) levels are to be determined as a function of the computed LOD.

LOD Computation Pseudocode

This section illustrates the LOD biasing and clamping computation in pseudocode, encompassing the steps described in the previous sections. The computation of the initial per-pixel LOD value *LOD* is not shown.

```
Bias:        S4.8
MinLod:      U4.8
MaxLod:      U4.8
Base:        U4.1
MIPCnt:      U4
SurfMinLod:  U4.8
ResMinLod:   U4.8


PerSampleMinLOD: float32

MinLod         = max(MinLod, PerSampleMinLOD)
AdjMaxLod      = min(MaxLod, MIPCnt)
AdjMinLod      = min(MinLod, MIPCnt)
AdjPR_minLOD   = ResMinLod - SurfMinLod
AdjMinLod      = max(AdjMinLod, AdjPR_minLOD)
Out_of_Bounds  = AdjPR_minLOD > MIPCnt

if ( sample_b )
    LOD += Bias + bias_parameter
else if ( sample_l or ld )
    LOD = Bias + lod_parameter
else
    LOD += Bias
```

Pseudocode

```
    PreClamp = LODPreClampMode != PRECLAMP_NONE
    if ( PreClamp )
        if ( PRECLAMP_D3D )
            LOD = min(LOD, AdjMaxLod)
            LOD = max(LOD, AdjMinLod)
        else
            LOD = min(LOD, MaxLod)
            LOD = max(LOD, MinLod)
```

```
MagMode = (LOD - Base <= 0)
```

Pseudocode

```
MagClampMipNone = LODClampMagnificationMode == MAG_CLAMP_MIPNONE
```

```
if ( (MagMode && MagClampMipNone) or MipFlt == None )
    LOD = 0
    LOD = min(LOD, ceil(AdjMaxLod))
    LOD = max(LOD, floor(AdjMinLod))
else if ( MipFlt == Nearest )
```

Pseudocode

```
LOD = min(LOD, ceil(AdjMaxLod))
LOD = max(LOD, floor(AdjMinLod))
```

```
    LOD += 0.5
    LOD = floor(LOD)
else
    // MipFlt = Linear
    LOD = min(LOD, AdjMaxLod)
    LOD = max(LOD, AdjMinLod)
    TriBeta = frac(LOD)
    LOD0 = floor(LOD)
    LOD1 = LOD0 + 1

if ( ! lod )  // "LOD" message type
    Lod += SurfMinLod
```

If Out_of_Bounds is true, LOD is set to zero and instead of sampling the surface the texels are replaced with zero in all channels, except for surface formats that don't contain alpha, for which the alpha channel is replaced with one. These texels then proceed through the rest of the pipeline.

Intra-Level Filtering Setup

Depending on whether the texture is being minified or magnified, the *MinFilter* or *MagFilter* state variable (respectively) is used to select the sampling filter to be used within a mip level (intra-level, as opposed to any inter-level filter). Note that for volume maps, this selection also applies to filtering between layers.

The processing at this stage is restricted to the selection of the filter type, computation of the number and texture map coordinates of the texture samples, and the computation of any required filter parameters. The filtering of the samples occurs later on in the Sampling Engine function.

The following table summarizes the intra-level filtering modes.

Sampler[]Min/MagFilter value	Description
MAPFILTER_NEAREST	Supported on all surface types. The texel nearest to the pixel's U,V,Q coordinate is read and output from the filter.
MAPFILTER_LINEAR	Not supported on buffer surfaces. The 2, 4, or 8 texels (depending on 1D, 2D/CUBE, or 3D surface, respectively) surrounding the pixel's U,V,Q coordinate are read and a linear filter is applied to produce a single filtered texel value.
MAPFILTER_ANISOTROPIC	Not supported on buffer or 3D surfaces. A projection of the pixel onto the texture map is generated and "subpixel" samples are taken along the major axis of the projection (center axis of the longer dimension). The outermost subpixels are weighted according to closeness to the edge of the projection, inner subpixels are weighted equally. Each subpixel samples a bilinear 2x2 of texels and the results are blended according to weights to produce a filtered texel value.
MAPFILTER_MONO	Supported only on 2D surfaces. This filter is only supported with the monochrome (MONO8) surface format. The monochrome texel block of the specified size surrounding the pixel is selected and filtered.

MAPFILTER_NEAREST

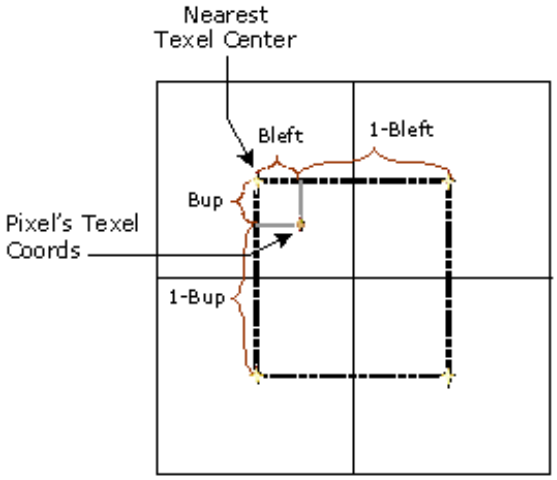
When the MAPFILTER_NEAREST is selected, the texel with coordinates nearest to the pixel's texture coordinate is selected and output as the single texel sample coordinates for the level. This is a form of "Point Sampling".

MAPFILTER_LINEAR

The following description indicates behavior of the MIPFILTER_LINEAR filter for 2D and CUBE surfaces. 1D and 3D surfaces follow a similar method but with a different number of dimensions available.

When the MAPFILTER_LINEAR filter is selected on a 2D surface, the 2x2 region of texels surrounding the pixel's texture coordinate are sampled and later bilinearly filtered. The filter weights each texel's contribution according to its distance from the pixel center. Texels further from the pixel center receive a smaller weight.

Bilinear Filter Sampling

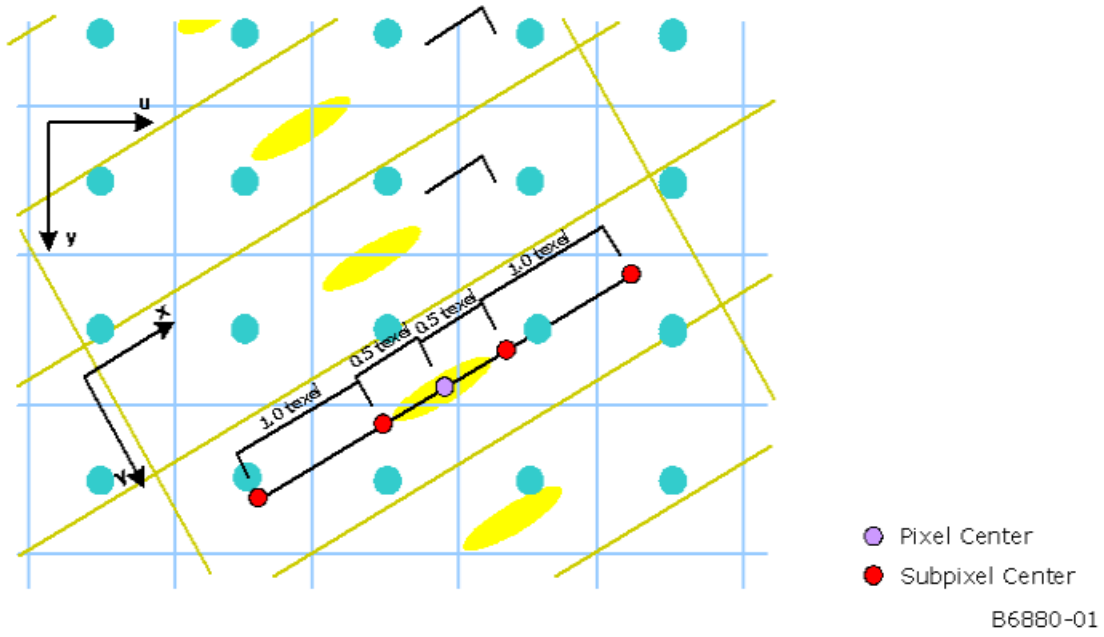


B6879-01

MAPFILTER_ANISOTROPIC

The MAPFILTER_ANISOTROPIC texture filter attempts to compensate for the anisotropic mapping of pixels into texture map space. A possibly non-square set of texel sample locations will be sampled and later filtered. The *MaxAnisotropy* state variable is used to select the maximum aspect ratio of the filter employed, up to 16:1.

The algorithm employed first computes the major and minor axes of the pixel projection onto the texture map. LOD is chosen based on the minor axis length in texel space. The anisotropic "ratio" is equal to the ratio between the major axis length and the minor axis length. The next larger even integer above the ratio determines the anisotropic number of "ways", which determines how many subpixels are chosen. A line along the major axis is determined, and "subpixels" are chosen along this line, spaced one texel apart, as shown in the diagram below. In this diagram, the texels are shown in light blue, and the pixels are in yellow.



Each subpixel samples a bilinear 2x2 around it just as if it was a single pixel. The result of each subpixel is then blended together using equal weights on all interior subpixels (not including the two endpoint subpixels). The endpoint subpixels have lesser weight, the value of which depends on how close the "ratio" is to the number of "ways". This is done to ensure continuous behavior in animation.

MAPFILTER_MONO

When the MAPFILTER_MONO filter is selected, a block of monochrome texels surrounding the pixel sample location are read and filtered using the kernel described below. The size of this block is controlled by **Monochrome Filter Height** and **Width** (referred to here as N_v and N_u, respectively) state. Filters from 1x1 to 7x7 are supported (not necessarily square).

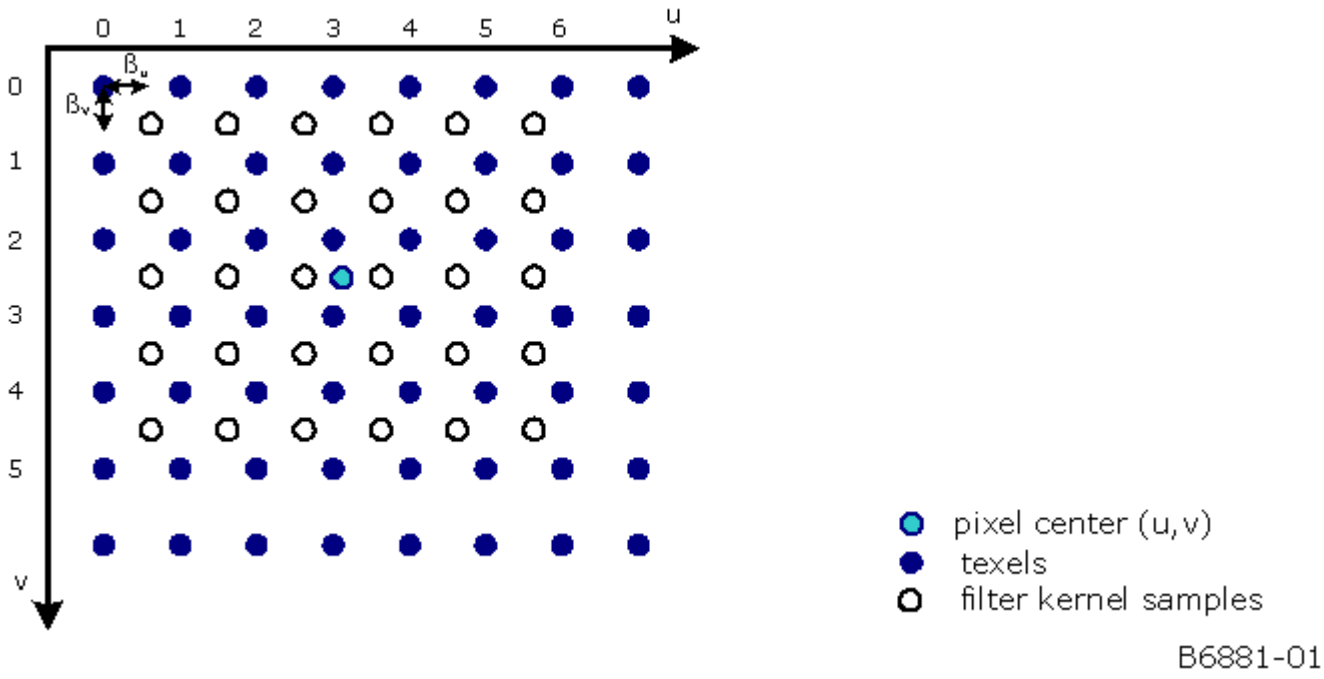
The figure below shows a 6x5 filter kernel as an example. The footprint of the filter (filter kernel samples) is equal to the size of the filter and the pixel center lies at the exact center of this footprint. The position of the upper left filter kernel sample (u_f, v_f) relative to the pixel center at (u, v) is given by the following:

$$u_f = u - \frac{N_u}{2}$$

$$v_f = v - \frac{N_v}{2}$$

b_u and b_v are the fractional parts of u_f and v_f, respectively. The integer parts select the upper left texel for the kernel filter, given here as $T_{0,0}$.

Sampling Using MAPFILTER_MONO



The formula for the final filter output F is given by the following. Since this is a monochrome filter, each texel value (T) is a single bit, and the output F is an intensity value that is replicated across the color and alpha channels.

$$S = \frac{1}{N_u * N_v}$$

$$F = \left[(1-\beta_u)(1-\beta_v)\sum_{i=0}^{N_u-1}\sum_{j=0}^{N_v-1} T_{i,j} + \beta_u(1-\beta_v)\sum_{i=1}^{N_u}\sum_{j=0}^{N_v-1} T_{i,j} + (1-\beta_u)\beta_v\sum_{i=0}^{N_u-1}\sum_{j=1}^{N_v} T_{i,j} + \beta_u\beta_v\sum_{i=1}^{N_u}\sum_{j=1}^{N_v} T_{i,j} \right] * S$$

Inter-Level Filtering Setup

The *MipFilter* state variable determines if and how texture mip maps are to be used and combined. The following table describes the various mip filter modes:

MipFilter Value	Description
MIPFILTER_NONE	Mipmapping is DISABLED. Apply a single filter on the highest resolution map available (after LOD clamping).
MIPFILTER_NEAREST	Choose the nearest mipmap level and apply a single filter to it. Here the biased LOD will be rounded to the nearest integer to obtain the desired miplevel. LOD Clamping may further restrict this miplevel selection.
MIPFILTER_LINEAR	Apply a filter on the two closest mip levels and linear blend the results using the distance between the computed LOD and the level LODs as the blend factor. Again, LOD Clamping may further restrict the selection of miplevels (and the blend factor between them).

When minifying and MIPFILTER_NEAREST is selected, the computed LOD is rounded to the nearest mip level.

When minifying and MIPFILTER_LINEAR is selected, the fractional bits of the computed LOD are used to generate an inter-level blend factor. The LOD is then truncated. The mip level selected by the truncated LOD, and the next higher (lower resolution) mip level are determined.

Regardless of *MipFilter* and the min/mag determination, all computed LOD values (two for MIPFILTER_LINEAR, otherwise one) are then unconditionally clamped to the range specified by the (integer bits of) *MinLOD* and *MaxLOD* state variables.

Texture Address Control

The *[TCX,TCY,TCZ]ControlMode* state variables control the access and/or generation of texel data when the specific texture coordinate component falls *outside* of the normalized texture map coordinate range [0,1).

The table below provides all the supported Address Control modes for each direction.

TC[X,Y,Z] Control	Operation
TEXCOORDMODE_CLAMP	Clamp to the texel value at the edge of the map.
TEXCOORDMODE_CLAMP_BORDER	Use the texture map's border color for any texel samples falling outside the map. The border color is specified via a pointer in SAMPLER_STATE.
TEXCOORDMODE_HALF_BORDER	Similar to CLAMP_BORDER except texels outside of the map are clamped to a value halfway between the edge texel and the border color.
TEXCOORDMODE_WRAP	Upon crossing an edge of the map, repeat at the other side of the map in the same dimension.
TEXCOORDMODE_CUBE	Only used for cube maps. Here texels from adjacent cube faces can be sampled along the edges of faces. This is considered the highest quality mode for cube environment maps.
TEXCOORDMODE_MIRROR	Similar to the wrap mode, though reverse direction through the map each time an edge is crossed. INVALID for use with unnormalized texture coordinates.

Separate controls are provided for texture TCX, TCY, TCZ coordinate components so, for example, the TCX coordinate can be wrapped while the TCY coordinate is clamped. Note that there are no controls provided for the TCW component as it is only used to scale the other 3 components before addressing modes are applied.

Programming Note	
Context:	Texture Address Control
TEXCOORDMODE_CUBE can only be used with SURFTYPE_CUBE	

Maximum Wraps/Mirrors

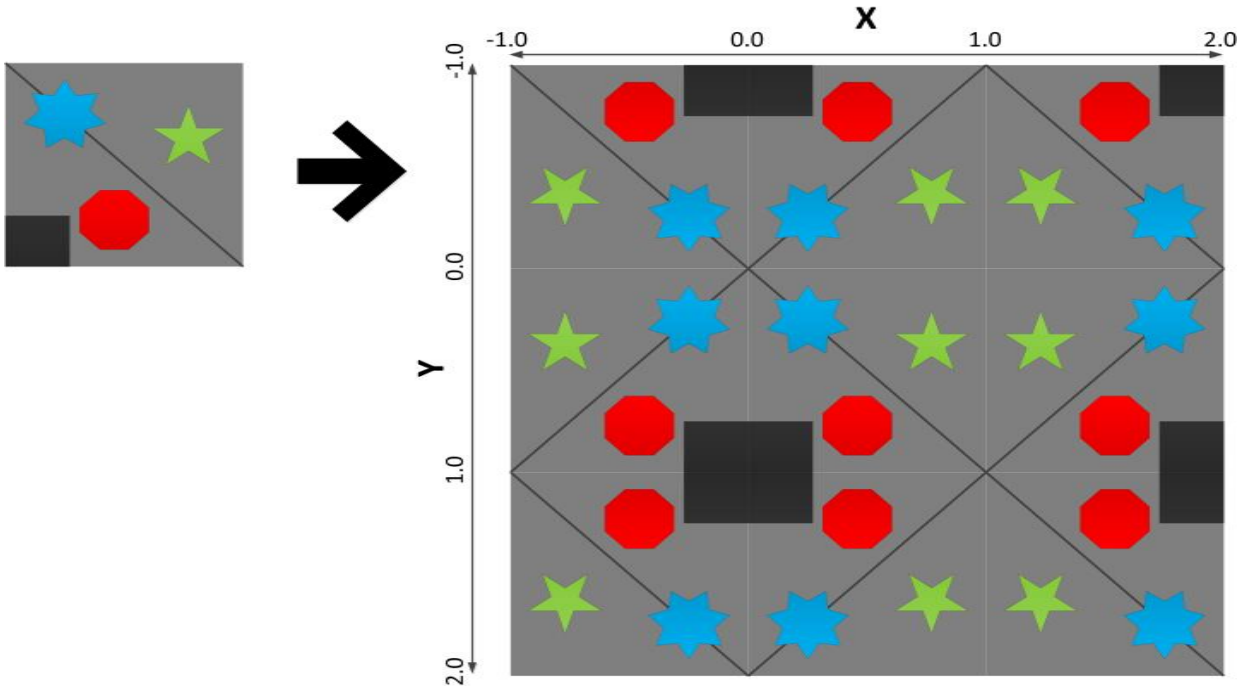
The number of map wraps on a given object is limited to 32. Going beyond this limit is legal, but may result in artifacts due to insufficient internal precision, especially evident with larger surfaces. Precision loss starts at the subtexel level (slight color inaccuracies) and eventually reaches the texel level (choosing the wrong texels for filtering).

Note: For **Wrap Shortest** mode, the setup kernel has already taken care of correctly interpolating the texture coordinates. Software needs to specify TEXCOORDMODE_WRAP mode for the sampler that is provided with wrap-shortest texture coordinates, or artifacts may be generated along map edges.

TEXCOORDMODE_MIRROR Mode

TEXCOORDMODE_MIRROR addressing mode is similar to Wrap mode, though here the base map is flipped at every integer junction. For example, for U values between 0 and 1, the texture is addressed normally, between 1 and 2 the texture is flipped (mirrored), between 2 and 3 the texture is normal again, and so on. The second row of pictures in the figure below indicate a map that is mirrored in one direction and then both directions. You can see that in the mirror mode every other integer map wrap the base map is mirrored in either direction.

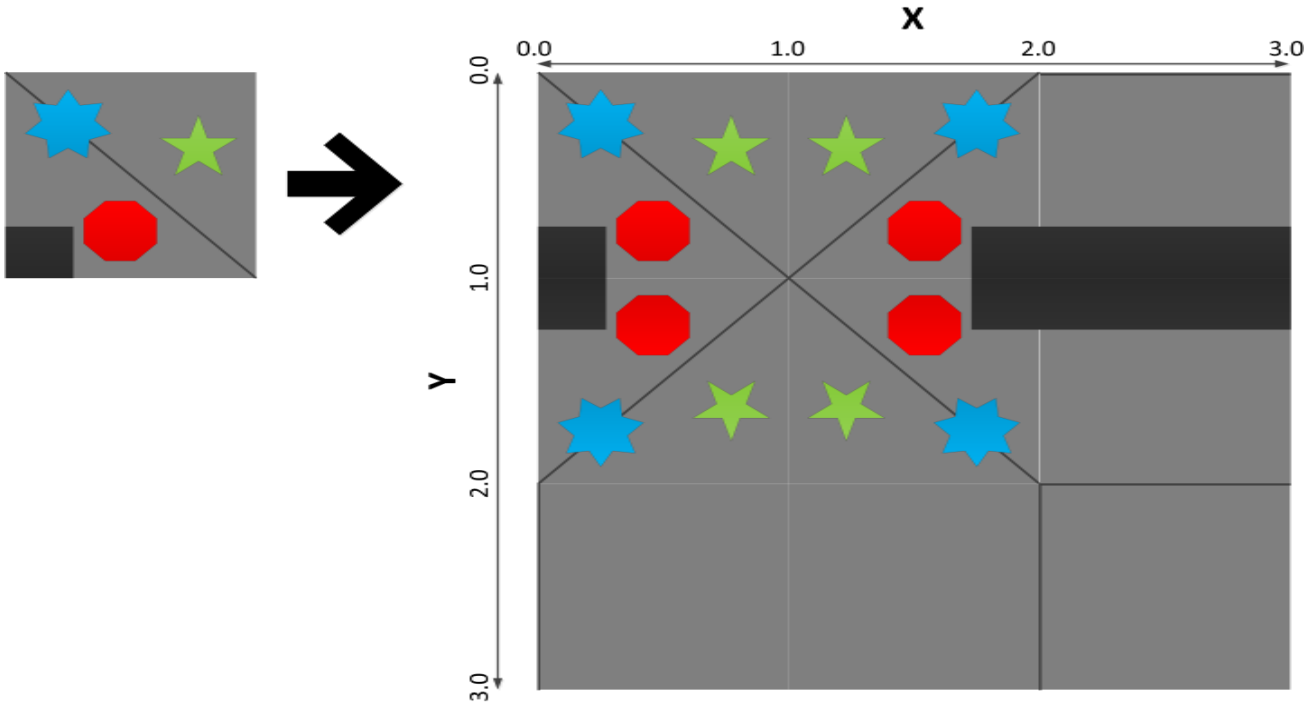
The example below shows how a simple 2D map with TEXCOORDMODE_MIRROR for both TCX and TCY is mapped.



TEXCOORDMODE_MIRROR_ONCE Mode

The TEXCOORDMODE_MIRROR_ONCE addressing mode is a combination of Mirror and Clamp modes. The absolute value of the texture coordinate component is first taken (thus mirroring about 0), and then the result is clamped to 1.0. The map is therefore mirrored once about the origin, and then clamped thereafter. This mode is used to reduce the storage required for symmetric maps.

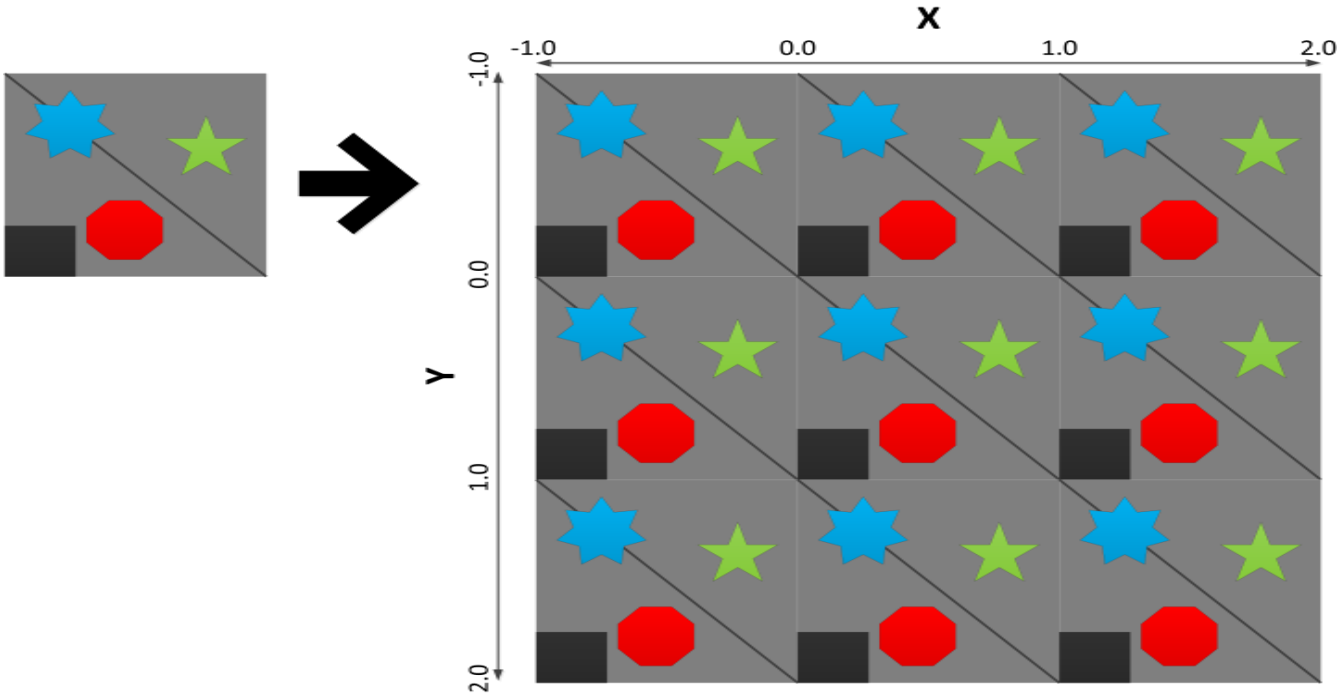
The example below shows how a simple 2D map with TEXCOORDMODE_MIRROR_ONCE for both TCX and TCY is mapped.



TEXCOORDMODE_WRAP Mode

In TEXCOORDMODE_WRAP addressing mode, the integer part of the texture coordinate is discarded, leaving only a fractional coordinate value. This results in the effect of the base map ([0,1)) being continuously repeated in all (axes-aligned) directions. Note that the interpolation between coordinate values 0.1 and 0.9 passes through 0.5 (as opposed to WrapShortest mode which interpolates through 0.0).

The example below shows how a simple 2D map with TEXCOORDMODE_WRAP for both TCX and TCY is mapped.

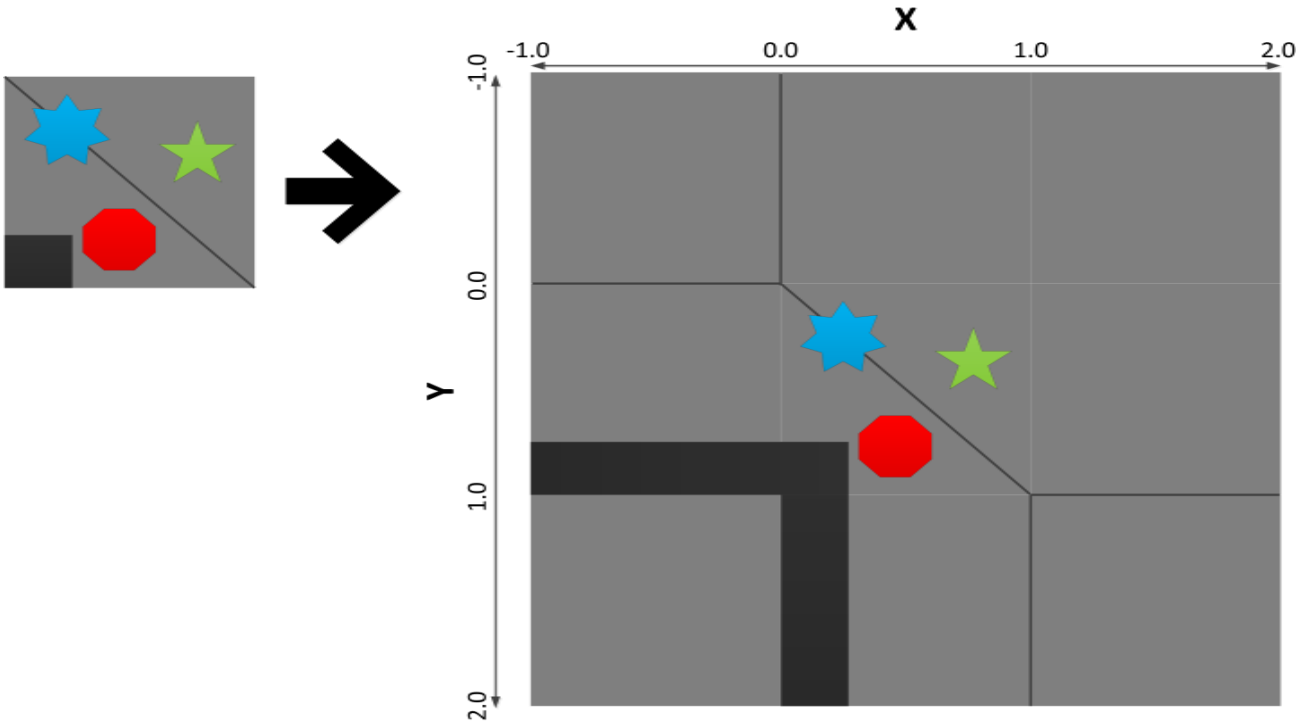


TEXCOORDMODE_CLAMP Mode

The TEXCOORDMODE_CLAMP addressing mode repeats the “edge” texel when the texture coordinate extends outside the [0,1) range of the base texture map. This is contrasted to TEXCOORDMODE_CLAMPBORDER mode which defines a separate texel value for off-map samples. TEXCOORDMODE_CLAMP is also supported for cube maps, where texture samples will only be obtained from the intersecting face (even along edges).

The figure below illustrates the effect of clamp mode. The base texture map is shown, along with a texture mapped object with texture coordinates extending outside of the base map region.

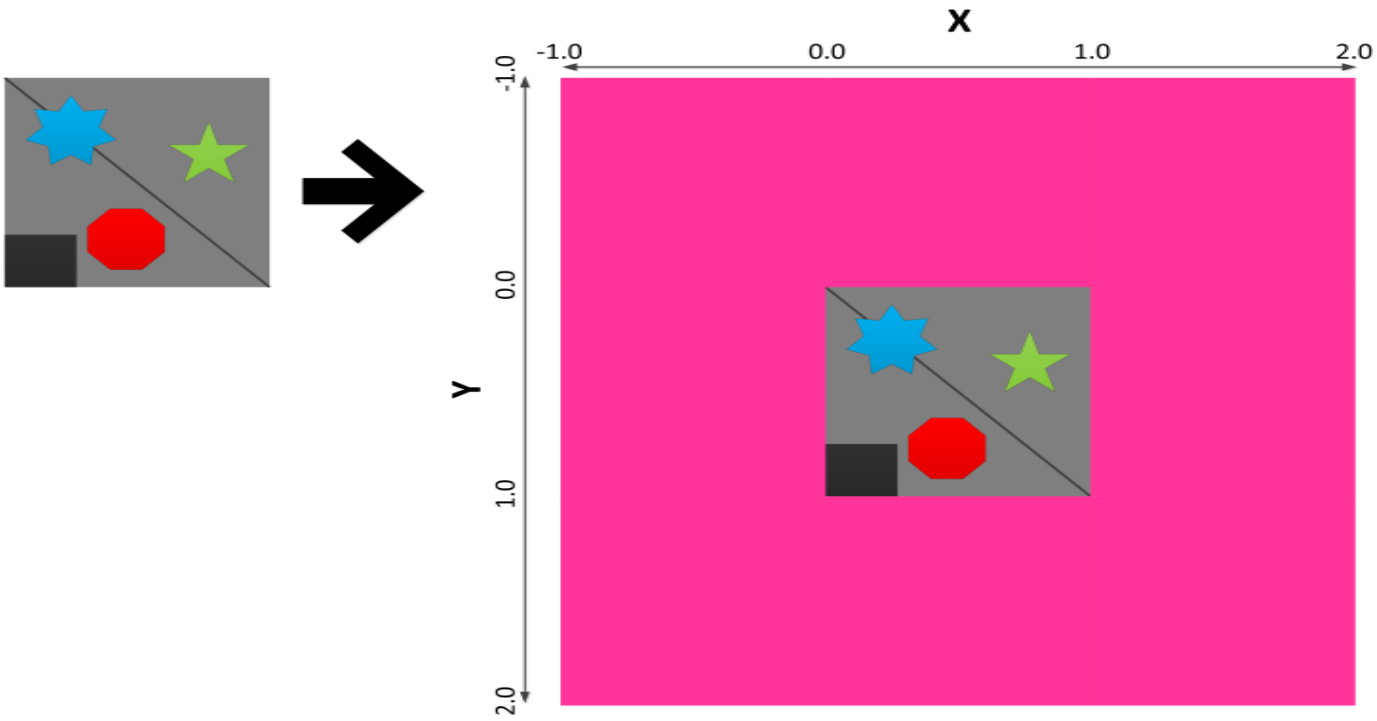
Texture Clamp Mode



TEXCOORDMODE_CLAMPBORDER Mode

For non-cube map textures, TEXCOORDMODE_CLAMPBORDER addressing mode specifies that the texture map's border value *BorderColor* is to be used for any texel samples that fall outside of the base map. The border color is specified via a pointer in SAMPLER_STATE.

The example below shows how a simple 2D map with TEXCOORDMODE_CLAMPBORDER for both TCX and TCY is mapped.



TEXCOORDMODE_HALF_BORDER Mode

For non-cube map textures, TEXCOORDMODE_HALF_BORDER addressing mode is similar to the TEXCOORDMODE_CLAMPBORDER in that it specifies that the texture map's border value *BorderColor*. However this value is blended with the edge texel color and used for any texel samples that fall outside of the base map. The border color is specified via a pointer in SAMPLER_STATE.

TEXCOORDMODE_CUBE Mode

For cube map textures TEXCOORDMODE_CUBE addressing mode can be set to allow inter-face filtering. When texel sample coordinates that extend beyond the selected cube face (e.g., due to intra-level filtering near a cube edge), the correct sample coordinates on the adjoining face will be computed. This will eliminate artifacts along the cube edges, though some artifacts at cube corners may still be present.

Texel Fetch

The Texel Fetch function of the Sampling Engine reads the texture map contents specified by the texture addresses associated with each texel sample. The texture data is read either directly from the memory-resident texture map, or from internal texture caches. The texture caches can be invalidated by the **Sampler Cache Invalidate** field of the MI_FLUSH instruction or via the **Read Cache Flush Enable** bit of PIPE_CONTROL. Except for consideration of coherency with CPU writes to textures and rendered textures, the texture cache does not affect the functional operation of the Sampling Engine pipeline.

When the surface format of a texture is defined as being a compressed surface, the Sampler will automatically decompress from the stored format into the appropriate [A]RGB values. The compressed texture storage formats and decompression algorithms can be found in the *Memory Data Formats* chapter. When the surface format of a texture is defined as being an index into the texture palette (format names includiong "Px"), the palette lookup of the index determines the appropriate RGB values.

Texel Chroma Keying

ChromaKey is a term used to describe a method of effectively removing or replacing a specific range of texel values from a map that is applied to a primitive, e.g., in order to define transparent regions in an RGB map. The Texel Chroma Keying function of the Sampling Engine pipeline conditionally tests texel samples against a "key" range, and takes certain actions if any texel samples are found to match the key.

Chroma Key Testing

ChromaKey refers to testing the texel sample components to see if they fall within a range of texel values, as defined by *ChromaKey[][High,Low]* state variables. If each component of a texel sample is found to lie within the respective (inclusive) range and ChromaKey is enabled, then an action will be taken to remove this contribution to the resulting texel stream output. Comparison is done separately on each of the channels and only if all 4 channels are within range the texel will be eliminated.

The Chroma Keying function is enabled on a per-sampler basis by the *ChromaKeyEnable* state variable.

The *ChromaKey[][High,Low]* state variables define the tested color range for a particular texture map.

Chroma Key Effects

There are two operations that can be performed to "remove" matching texel samples from the image. The *ChromaKeyEnable* state variable must first enable the chroma key function. The *ChromaKeyMode* state variable then specifies which operation to perform on a per-sampler basis.

The *ChromaKeyMode* state variable has the following two possible values:

KEYFILTER_KILL_ON_ANY_MATCH: Kill the pixel if any contributing texel sample matches the key.

KEYFILTER_REPLACE_BLACK: Here the sample is replaced with (0,0,0,0).

The Kill Pixel operation has an effect on a pixel only if the associated sampler is referenced by a sample instruction in the pixel shader program. If the sampler is not referenced, the chroma key compare is not done and pixels cannot be killed based on it.

Shadow Prefilter Compare

When a *sample_c* message type is processed, a special shadow-mapping precomparison is performed on the texture sample values prior to filtering. Specifically, each texture sample value is compared to the "ref" component of the input message, using a compare function selected by *ShadowFunction*, and described in the table below. Note that only single-channel texel formats are supported for shadow mapping, and so there is no specific color channel on which the comparison occurs.

ShadowFunction	Result
PREFILTEROP_ALWAYS	0.0
PREFILTEROP_NEVER	1.0
PREFILTEROP_LESS	(texel < ref) ? 0.0 : 1.0
PREFILTEROP_EQUAL	(texel == ref) ? 0.0 : 1.0
PREFILTEROP_LEQUAL	(texel <= ref) ? 0.0 : 1.0
PREFILTEROP_GREATER	(texel > ref) ? 0.0 : 1.0
PREFILTEROP_NOTEQUAL	(texel != ref) ? 0.0 : 1.0
PREFILTEROP_GEQUAL	(texel >= ref) ? 0.0 : 1.0

The binary result of each comparison is fed into the subsequent texture filter operation (in place of the texel's value which would normally be used).

Software is responsible for programming the "ref" component of the input message such that it approximates the same distance metric programmed in the texture map (e.g., distance from a specific light to the object pixel). In this way, the comparison function can be used to generate "in shadow" status for each texture sample, and the filtering operation can be used to provide soft shadow edges.

Texel Filtering

The Texel Filtering function of the Sampling Engine performs any required filtering of multiple texel values on and possibly between texture map layers and levels. The output of this function is a single texel color value.

The state variables *MinFilter*, *MagFilter*, and *MipFilter* are used to control the filtering of texel values. The *MipFilter* state variable specifies how many mipmap levels are included in the filter, and how the results of any filtering on these separate levels are combined to produce a final texel color. The *MinFilter* and *MagFilter* state variables specify how texel samples are filtered within a level.

Texel Color Gamma Linearization

This function is supported to allow pre-gamma-corrected texel RGB (not A) colors to be mapped back into linear (gamma=1.0) gamma space prior to (possible) blending with, and writing to the Color Buffer. This permits higher quality image blending by performing the blending on colors in linear gamma space.

This function is enabled on a per-texture basis by use of a surface format with "_SRGB" in its name. If enabled, the pre-filtered texel RGB color to be converted to gamma=1.0 space by applying a ^(2.4) exponential function.

Multisampled Surface Behavior

Multisampled surfaces are sampled using a dedicated point-sample message **ld2dms** or **l2dms_w**. These message contain Si (sample index) which is used to sample from a specific subsample plane. Each texel of a multi-sampled surface has 1,2,4,8 or 16 "subsamples" per texel.

The sampleinfo message returns specific parameters associated with a multisample surface such as the number of subsamples per texel. The resinfo message returns the height, width, depth (in units of *pixels*, not samples), and MIP count of the surface .

Multisampled surfaces typically have an associated MCS control surface which contains information about which indicates which sample index contains the color data for a specific subample. The control surface can be accessed using the **ld_mcs** message, which returns the information for all subsamples for a given texel.

From an optimization perspective there are two different approaches which can be used to resolver a multisampled surface:

A shader can fetch ld_mcs to determine which subsamples need to be fetched using ld2dms or ld2dms_w. In general, it is expected that most subsamples will reference Sample index zero for it's color by default. It may be optimal for the shader to always fetch to sample index 0 unconditionally, and only fetch to other sample indices based on the fetched MCS. Resolve can then be done with the minimumal number of point-sample operations.

Alternatively, a shader can unconditionally fetch all subsamples to minimize dependencies. In this case the MCS data returned from the ld_mcs is simply passsed to the ld2dms message.

Multisample Control Surface

Four messages have been defined for the sampling engine, *ld_mcs*, *ld2dms*, *ld2dms_w* and *ld2dss*. A pixel shader kernel sampling from an multisampled surface using an MCS must first sample from the MCS surface using the *ld_mcs* message. This message behaves like the *ld* message, except that the surface is defined by the MCS fields of SURFACE_STATE rather than the normal fields. The surface format is effectively R8_UINT for 4x surfaces, R32_UINT for 8x surfaces, and two R32_UNIT for x16 surfaces thus data is returned in unsigned integer format. Following the *ld_mcs*, the kernel issues a *ld2dms* or ld2dms_w message to sample the surface itself. The integer value from the MCS surface is delivered in the mcs parameter of this messages as the sample index.

Since *sample* is no longer supported on multisampled surfaces, the multisample resolve must be done using *ld2dms or ld2dms_w*. For surfaces with **Multisampled Surface Storage Format** set to MSFMT_MSS and **MCS Enable** set to enabled, an optimization is available to enable higher performance for compressed pixels. The *ld2dss* message can be used to sample from a particular sample slice on the surface. By examining the MCS value, software can determine which sample slices to sample from. A simple optimization with probable large return in performance is to compare the MCS value to zero (indicating all samples are on sample slice 0), and sample only from sample slice 0 using *ld2dss* if MCS is zero. Sample slice 0 is the pixel color in this case. If MCS is not zero, each sample is then obtained using *ld2dms* messages and the results are averaged in the kernel after being returned. Refer to the multisample storage format in the GPU Overview volume for more details.

Quilted Textures]

Quilted textures allow very large 2D textures, up to 512K x 512K texels. Rather than simply increasing the height and width ranges, a quilted texture is stored in memory as a 2D array. (Refer to the GPU Overview volume for details on 2D array storage.) Each array slice has a maximum size of 16K x 16K texels, and these slices are referred to as “quilt slices.” Two fields in SURFACE_STATE, Quilt Width and Quilt Height, define the size of the quilt in units of quilt slices. A surface is defined as a quilted texture if either one of the above fields are set to a value other than 1. Quilted textures are supported with all address control modes, and with all sample* messages that support 2D arrays and use the SURFACE_STATE “for most messages.” In general, whether the surface is quilted or not is orthogonal to the message definition.

MIP maps are supported with quilted textures, with the quilt slices being reduced by a factor of 2 in each dimension for each increment in LOD. The minimum size MIP is where the quilt slice is a 1x1. Effective QuiltWidth and QuiltHeight are not reduced with increasing LOD which would cause the smallest mip to be QuiltWidth x QuiltHeight rather than the typical 1x1.

Refer to the Quilted Textures section of Common Surface Formats, Surface Layout and Tiling for details on how quilted textures are mapped into the 2D array storage layout.

The “u” and “v” parameters must be normalized, interpreted as coordinates across the entire quiltW, thus software doesn’t determine which quilt slices need to be accessed.

A new set of messages, called “SIMD8D”, is introduced to support double precision “u” and “v” parameters. The use of SIMD8D, while motivated by quilted textures, is orthogonal to them from the architecture standpoint. Both quilted and non-quilted textures can be accessed using single precision or double precision “u”/”v” parameters. Hardware will use the precision delivered to access the texture, with artifacts resulting if insufficient “u”/”v” precision is used.ves farther away from the base location more bits of beta will be lost.

Quilted textures can only be supported on tiled surfaces (e.g. TileY, TileYs, TileYf).

State

State
SW Generated BINDING_TABLE_STATE
HW Generated BINDING_TABLE_STATE

For **SAMPLER_STATE** for **Sample_8X8** see 3D-Media-GPGPU Engine > Shared Functions > Media Sampler > Sample_8x8 State > SAMPLER_STATE

The 3D sampler uses both surface state objects (RENDER_SURFACE_STATE) as well as sampler state objects (SAMPLER_STATE). These objects are cached locally in the sampler state cache

for improved performance as it is assumed that many sampler messages will utilize the same surface and sampler states.

Programming Note	
Context:	Out of Bounds Handling
If a pointer to sampler or surface state goes beyond the end of the sampler or surface state buffer (as defined by the associated size field of the STATE_BASE_ADDRESS command) the sampler will force the address offset to cache-line 0 from the defined Base Address. The result of this state fetch is undefined and depends on how the state buffer has been populated.	

Surface State Fetch

Surface state is fetched from system memory using a Binding Table Pointer (**BTP**). The **BTP** is a 16-bit value provided by the command stream (not directly by the shader) which determines the binding-table to be used. An 8-bit Binding Table Index (**BTI**) is then provided by the shader via the message descriptor, which indicates the offset into the Binding Table. The BTP and BTI are relative to the **Surface State Base Address** and the binding table itself resides in system memory. The contents of the Binding Table is a list of pointers to surface state objects. The pointer from the Binding Table is also relative to the **Sampler State Base Address,** and points directly to a 256-bit **RENDER_SURFACE_STATE** object which sampler will fetch and store in its internal state cache.

Bindless Surface State Fetch

The sampler supports a "Bindless" surface model. Bindless refers to the fact that a Binding table is not required for this mode. Instead, the shader sets the BTI to 253 in the descriptor to indicate Bindless and provides a 20-bit **Surface State Offset (SSO)** from the **Bindless Surface State Base Address .** This SSO directly selects a specific surface state object which is cached by the 3D Sampler in local State Cache. Both bindless and non-bindless (legacy) modes can be operated at the same time by the sampler.

Sampler State Fetch

SAMPLER_STATE objects are fetched independely of surface state and cached locally in the 3D sampler independently (there may one or more SAMPLER_STATE objects associates with one or more

RENDER_SURFACE_STATE objects). The sampler state is fetched using the **Sampler State Pointer (SSP)** which is provided either in the message header or directly from the command stream (message headers are not required). The **SSP** is an offset relative to the **Dynamic_State_Base_Address** and selects a table of 16 sampler states. The 4-bit **Sampler Index** (**SI**) in the message descriptor is used to select the specific **SAMPLER_STATE** object to be fetched from system memory and cached locally in the 3D sampler.

Bindless Sampler State

The sampler supports a "Bindless" sampler model. Bindless in this case does not actually refer to the lack of a Binding table since the legacy Sampler State model also did not have a Binding Table. However, the mechanism is similar to bindless surfaces in that the pointer provided directly selects a SAMPLER_STATE object. The sampler uses the same **Sampler State Pointer (SSP)**, but it is relative to the **Bindless_Surface_State_Base_Address** rather than the Dynamic_State_Base_Addr. Bindless Samplers can only be used in conjunction with Bindless Surfaces. The Sampler Index (SI) in the message descriptor is not used, and can be set to 0. The SAMPLER_STATE object is cached locally in the 3D sampler.

State Caching

As mentioned above, the 3D Sampler allows for automatic caching of **RENDER_SURFACE_STATE** objects and **SAMPLER_STATE** objects to provide higher performance. Coherency with system memory in the state cache, like the texture cache is handled partially by software. It is expected that the command stream or shader will issue Cache Flush operation or Cache_Flush sampler message to ensure that the L1 cache remains coherent with system memory.

Programming Note	
Context:	State Cache Coherency
Whenever the value of the Dynamic_State_Base_Addr, Surface_State_Base_Addr are altered, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory. Whenever the RENDER_SURFACE_STATE object in memory pointed to by the **Binding Table Pointer (BTP)** and **Binding Table Index (BTI)** is modified or SAMPLER_STATE object pointed to by the **Sampler State Pointer (SSP)** and **Sampler Index (SI)** is modified, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory.	

Programming Note	
Context:	Bindless Surface State Cache Coherency
Whenever the value of the Bindless_Surface_State_Base_Addr is altered, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory. Whenever the RENDER_SURFACE_STATE object in memory pointed to by the **Surface State Offset (SSO)** is modified, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory.	

SURFACE_STATE

The surface state is stored as individual elements, each with its own pointer in the binding table or its own entry in a memory heap in memory. Each surface state element is aligned to a 32-byte boundary.

Surface state defines the state needed for the following objects:

- texture maps (1D, 2D, 3D, cube) read by the sampling engine
- buffers read by the sampling engine
- constant buffers read by the data cache via the data port
- render targets read/written by the render cache via the data port
- streamed vertex buffer output written by the render cache via the data port
- media surfaces read from the texture cache or render cache via the data port
- media surfaces written to the render cache via the data port

The surface state definition can be found in the following section:

Section
RENDER_SURFACE_STATE

Surface Formats

The RENDER_SURFACE_STATE contains a 9-bit field called **Surface Format**, which defines the exact format of the surface being sampled. The definition of the encodings for each supported format, including compressed formats can be found in the following section:

SURFACE_FORMAT [All]

For ASTC formats, the **ASTC Enable** bit in the RENDER_SURFACE_STATE must be set to 1. When set, the definition of the 9-bit **Surface Format** changes. The following table describes all supported formats for block-based ASTC textures.

SURFACE_FORMAT for All ASTC Formats

Value	[8] LDR/Full [7] 2D/3D [6] U8srgb /FLT16	Width 2D [5:3] 3D [5:4]	Height 2D [2:0] 3D [3:2]	Depth 2D: N/A 3D: [1:0]	Binary Form	Name	(BPE)
000h	000	0	0		000 000 000	ASTC_LDR_2D_4x4_U8sRGB	8.00
008h	000	1	0		000 001 000	ASTC_LDR_2D_5x4_U8sRGB	6.40
009h	000	1	1		000 001 001	ASTC_LDR_2D_5x5_ U8sRGB	5.12

Value	[8] LDR/Full [7] 2D/3D [6] U8srgb /FLT16	Width 2D [5:3] 3D [5:4]	Height 2D [2:0] 3D [3:2]	Depth 2D: N/A 3D: [1:0]	Binary Form	Name	(BPE)
011h	000	2	1		000 010 001	ASTC_LDR_2D_6x5_ U8sRGB	4.27
012h	000	2	2		000 010 010	ASTC_LDR_2D_6x6_ U8sRGB	3.56
021h	000	4	1		000 100 001	ASTC_LDR_2D_8x5_ U8sRGB	3.20
022h	000	4	2		000 100 010	ASTC_LDR_2D_8x6_ U8sRGB	2.67
031h	000	6	1		000 110 001	ASTC_LDR_2D_10x5_ U8sRGB	2.56
032h	000	6	2		000 110 010	ASTC_LDR_2D_10x6_ U8sRGB	2.13
024h	000	4	4		000 100 100	ASTC_LDR_2D_8x8_ U8sRGB	2.00
034h	000	6	4		000 110 100	ASTC_LDR_2D_10x8_ U8sRGB	1.60
036h	000	6	6		000 110 110	ASTC_LDR_2D_10x10_ U8sRGB	1.28
03eh	000	7	6		000 111 110	ASTC_LDR_2D_12x10_ U8sRGB	1.07
03fh	000	7	7		000 111 111	ASTC_LDR_2D_12x12_ U8sRGB	0.89
040h	001	0	0		001 000 000	ASTC_LDR_2D_4x4_FLT16	8.00
048h	001	1	0		001 001 000	ASTC_LDR_2D_5x4_FLT16	6.40
049h	001	1	1		001 001 001	ASTC_LDR_2D_5x5_FLT16	5.12
051h	001	2	1		001 010 001	ASTC_LDR_2D_6x5_FLT16	4.27
052h	001	2	2		001 010 010	ASTC_LDR_2D_6x6_FLT16	3.56
061h	001	4	1		001 100	ASTC_LDR_2D_8x5_FLT16	3.20

Value	[8] LDR/Full [7] 2D/3D [6] U8srgb /FLT16	Width 2D [5:3] 3D [5:4]	Height 2D [2:0] 3D [3:2]	Depth 2D: N/A 3D: [1:0]	Binary Form	Name	(BPE)
					001		
062h	001	4	2		001 100 010	ASTC_LDR_2D_8x6_FLT16	2.67
071h	001	6	1		001 110 001	ASTC_LDR_2D_10x5_FLT16	2.56
072h	001	6	2		001 110 010	ASTC_LDR_2D_10x6_FLT16	2.13
064h	001	4	4		001 100 100	ASTC_LDR_2D_8x8_FLT16	2.00
074h	001	6	4		001 110 100	ASTC_LDR_2D_10x8_FLT16	1.60
076h	001	6	6		001 110 110	ASTC_LDR_2D_10x10_FLT16	1.28
07eh	001	7	6		001 111 110	ASTC_LDR_2D_12x10_FLT16	1.07
07fh	001	7	7		001 111 111	ASTC_LDR_2D_12x12_FLT16	0.89
140h	101	0	0		101 000 000	ASTC_FULL_2D_4x4_FLT16	8.00
148h	101	1	0		101 001 000	ASTC_FULL_2D_5x4_FLT16	6.40
149h	101	1	1		101 001 001	ASTC_FULL_2D_5x5_FLT16	5.12
151h	101	2	1		101 010 001	ASTC_FULL_2D_6x5_FLT16	4.27
152h	101	2	2		101 010 010	ASTC_FULL_2D_6x6_FLT16	3.56
161h	101	4	1		101 100 001	ASTC_FULL_2D_8x5_FLT16	3.20
162h	101	4	2		101 100 010	ASTC_FULL_2D_8x6_FLT16	2.67
171h	101	6	1		101 110 001	ASTC_FULL_2D_10x5_FLT16	2.56

Value	[8] LDR/Full [7] 2D/3D [6] U8srgb /FLT16	Width 2D [5:3] 3D [5:4]	Height 2D [2:0] 3D [3:2]	Depth 2D: N/A 3D: [1:0]	Binary Form	Name	(BPE)
172h	101	6	2		101 110 010	ASTC_FULL_2D_10x6_FLT16	2.13
164h	100	4	4		101 100 100	ASTC_FULL_2D_8x8_FLT16	2.00
174h	101	6	4		101 110 100	ASTC_FULL_2D_10x8_FLT16	1.60
176h	101	6	6		101 110 110	ASTC_FULL_2D_10x10_FLT16	1.28
17eh	101	7	6		101 111 110	ASTC_FULL_2D_12x10_FLT16	1.07
17fh	101	7	7		101 111 111	ASTC_FULL_2D_12x12_FLT16	0.89

Sampler Output Channel Mapping

The following table indicates the mapping of the channels from the surface to the channels output from the sampling engine. Formats with all four channels (R/G/B/A) in their name map each surface channel to the corresponding output, thus those formats are not shown in this table.

Programming Note	
Context:	Luminance, Intensity and Transparency Formats
Luminance, Intensity and Opacity formats can be more efficiently supported with higher performance by mapping them to their equivalent RGB format and programming the Shader Channel selects in the RENDER_SURFACE_STATE. See SW Performance Hints for a detailed description of this useage model.	

There are further restrictions listed after the table below on the use of specific surfaces.

Some formats are supported only in DX10/OGL **Border Color Mode**. Those formats have only that mode indicated. Formats that behave the same way in both **Border Color Modes** are indicated by that column being blank. See the programming notes below the following table for more details on how to support these surfaces.

Surface Format Name	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A	Border Color Mode	R	G	B	A
R32G32B32A32_FLOAT				R	G	B	A					
R32G32B32A32_SINT			DX10/OGL	R	G	B	A					
R32G32B32A32_UINT			DX10/OGL	R	G	B	A					
R32G32B32X32_FLOAT				R	G	B	1.0					
R32G32B32_FLOAT				R	G	B	1.0					
R32G32B32_SINT			DX10/OGL	R	G	B	1.0					
R32G32B32_UINT			DX10/OGL	R	G	B	1.0					
R16G16B16A16_SNORM				R	G	B	A					
R16G16B16A16_SINT			DX10/OGL	R	G	B	A					
R16G16B16A16_UINT			DX10/OGL	R	G	B	A					
R16G16B16A16_FLOAT				R	G	B	A					
R32G32_FLOAT			DX10/OGL	R	G	0.0	1.0		R	G	1.0	1.0
R32G32_SINT			DX10/OGL	R	G	0.0	1.0					
R32G32_UINT			DX10/OGL	R	G	0.0	1.0					
R32_FLOAT_X8X24_TYPELESS	Yes		DX10/OGL	R	0.0	0.0	1.0					
X32_TYPELESS_G8X24_UINT			DX10/OGL	0.0	G	0.0	1.0					
L32A32_FLOAT			DX10/OGL	L	L	L	A					
R16G16B16X16_UNORM				R	G	B	1.0					
R16G16B16X16_FLOAT				R	G	B	1.0					
A32X32_FLOAT				0.0	0.0	0.0	A					
L32X32_FLOAT				L	L	L	1.0					
I32X32_FLOAT				I	I	I	I					
B8G8R8A8_UNORM		Yes		R	G	B	A					
B8G8R8A8_UNORM_SRGB				R	G	B	A					
R10G10B10A2_UNORM				R	G	B	A					
R10G10B10A2_UNORM_SRGB				R	G	B	A					
R10G10B10A2_UINT			DX10/OGL	R	G	B	A					
R10G10B10_SNORM_A2_UNORM				R	G	B	A					
R8G8B8A8_UNORM				R	G	B	A					
R8G8B8A8_UNORM_SRGB				R	G	B	A					
R8G8B8A8_SNORM				R	G	B	A					

Surface Format Name	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A	Border Color Mode	R	G	B	A
R8G8B8A8_SINT			DX10/OGL	R	G	B	A					
R8G8B8A8_UINT			DX10/OGL	R	G	B	A					
R16G16_SNORM			DX10/OGL	R	G	0.0	1.0		R	G	1.0	1.0
R16G16_SINT			DX10/OGL	R	G	0.0	1.0					
R16G16_UINT			DX10/OGL	R	G	0.0	1.0					
R16G16_FLOAT			DX10/OGL	R	G	0.0	1.0	DX9	R	G	1.0	1.0
B10G10R10A2_UNORM				R	G	B	A					
B10G10R10A2_UNORM_SRGB				R	G	B	A					
R11G11B10_FLOAT				R	G	B	1.0					
R32_SINT			DX10/OGL	R	0.0	0.0	1.0					
R32_UINT			DX10/OGL	R	0.0	0.0	1.0					
R32_FLOAT	Yes		DX10/OGL	R	0.0	0.0	1.0		R	1.0	1.0	1.0
R24_UNORM_X8_TYPELESS	Yes		DX10/OGL	R	0.0	0.0	1.0					
X24_TYPELESS_G8_UINT			DX10/OGL	0.0	G	0.0	1.0					
L16A16_UNORM				L	L	L	A					
I24X8_UNORM	Yes			I	I	I	I					
L24X8_UNORM	Yes			L	L	L	1.0					
A24X8_UNORM	Yes			0.0	0.0	0.0	A					
I32_FLOAT	Yes			I	I	I	I					
L32_FLOAT	Yes			L	L	L	1.0					
A32_FLOAT	Yes			0.0	0.0	0.0	A					
B8G8R8X8_UNORM		Yes		R	G	B	1.0					
B8G8R8X8_UNORM_SRGB				R	G	B	1.0					
R8G8B8X8_UNORM				R	G	B	1.0					
R8G8B8X8_UNORM_SRGB				R	G	B	1.0					
R9G9B9E5_SHAREDEXP				R	G	B	1.0					
B10G10R10X2_UNORM				R	G	B	1.0					
L16A16_FLOAT				L	L	L	A					
B5G6R5_UNORM		Yes		R	G	B	1.0					
B5G6R5_UNORM_SRGB				R	G	B	1.0					
B5G5R5A1_UNORM		Yes		R	G	B	A					
B5G5R5A1_UNORM_SRGB				R	G	B	A					
B4G4R4A4_UNORM		Yes		R	G	B	A					
B4G4R4A4_UNORM_SRGB				R	G	B	A					

Surface Format Name	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A	Border Color Mode	R	G	B	A
R8G8_UNORM			DX10/OGL	R	G	0.0	1.0	DX9	R	G	1.0	1.0
R8G8_SNORM		Yes	DX10/OGL	R	G	0.0	1.0	DX9	R	G	1.0	1.0
R8G8_SINT			DX10/OGL	R	G	0.0	1.0					
R8G8_UINT			DX10/OGL	R	G	0.0	1.0					
R16_SNORM			DX10/OGL	R	0.0	0.0	1.0					
R16_SINT			DX10/OGL	R	0.0	0.0	1.0					
R16_UINT			DX10/OGL	R	0.0	0.0	1.0					
R16_FLOAT			DX10/OGL	R	0.0	0.0	1.0		R	1.0	1.0	1.0
A8P8_UNORM_PALETTE0				R	G	B	A					
A8P8_UNORM_PALETTE1				R	G	B	A					
I16_UNORM	Yes			I	I	I	I					
L16_UNORM	Yes			L	L	L	1.0					
A16_UNORM	Yes			0.0	0.0	0.0	A					
L8A8_UNORM		Yes		L	L	L	A					
I16_FLOAT	Yes			I	I	I	I					
L16_FLOAT	Yes			L	L	L	1.0					
A16_FLOAT	Yes			0.0	0.0	0.0	A					
L8A8_UNORM_SRGB				L	L	L	A					
R5G5_SNORM_B6_UNORM		Yes		R	G	B	1.0					
P8A8_UNORM_PALETTE0				R	G	B	A					
P8A8_UNORM_PALETTE1				R	G	B	A					
A1B5G5R5_UNORM				R	G	B	A					
A4B4G4R4_UNORM				R	G	B	A					
R8_UNORM		Yes	DX10/OGL	R	0.0	0.0	1.0					
R8_SNORM			DX10/OGL	R	0.0	0.0	1.0					
R8_SINT			DX10/OGL	R	0.0	0.0	1.0					
R8_UINT			DX10/OGL	R	0.0	0.0	1.0					
A8_UNORM		Yes		0.0	0.0	0.0	A					
I8_UNORM				I	I	I	I					
L8_UNORM		Yes		L	L	L	1.0					
Y8_UNORM		Yes		Y	Y	Y	1.0					
P4A4_UNORM_PALETTE0				R	G	B	A					
A4P4_UNORM_PALETTE0				R	G	B	A					
P8_UNORM_PALETTE0				R	G	B	A					

Surface Format Name	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A	Border Color Mode	R	G	B	A
L8_UNORM_SRGB				L	L	L	1.0					
P8_UNORM_PALETTE1				R	G	B	A					
P4A4_UNORM_PALETTE1				R	G	B	A					
A4P4_UNORM_PALETTE1				R	G	B	A					
DXT1_RGB_SRGB				R	G	B	1.0					
R1_UNORM				R	0.0	0.0	1.0					
YCRCB_NORMAL		Yes		Cr	Y	Cb	1.0					
YCRCB_SWAPUVY		Yes		Cr	Y	Cb	1.0					
P2_UNORM_PALETTE0				R	G	B	A					
P2_UNORM_PALETTE1				R	G	B	A					
BC1_UNORM		Yes		R	G	B	A					
BC2_UNORM		Yes		R	G	B	A					
BC3_UNORM		Yes		R	G	B	A					
BC4_UNORM			DX10/OGL	R	0.0	0.0	1.0					
BC5_UNORM			DX10/OGL	R	G	0.0	1.0					
BC1_UNORM_SRGB				R	G	B	A					
BC2_UNORM_SRGB				R	G	B	A					
BC3_UNORM_SRGB				R	G	B	A					
MONO8				N/A	N/A	N/A	N/A					
YCRCB_SWAPUV				Cr	Y	Cb	1.0					
YCRCB_SWAPY				Cr	Y	Cb	1.0					
DXT1_RGB				R	G	B	1.0					
FXT1				R	G	B	A					
R8G8B8_UNORM				R	G	B	1.0					
R8G8B8_SNORM				R	G	B	1.0					
BC4_SNORM			DX10/OGL	R	0.0	0.0	1.0					
BC5_SNORM			DX10/OGL	R	G	0.0	1.0					
R16G16B16_FLOAT				R	G	B	1.0					
R16G16B16_UNORM				R	G	B	1.0					
R16G16B16_SNORM				R	G	B	1.0					
BC6H_SF16				R	G	B	1.0*					
BC7_UNORM				R	G	B	A					
BC7_UNORM_SRGB				R	G	B	A					
BC6H_UF16				R	G	B	1.0*					

Surface Format Name	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A	Border Color Mode	R	G	B	A
PLANAR_420_8				Cr	Y	Cb	1.0					
R8G8B8_UNORM_SRGB				R	G	B	1.0					
ETC1_RGB8				R	G	B	1.0					
ETC2_RGB8				R	G	B	1.0					
EAC_R11				R	0.0	0.0	1.0					
EAC_RG11				R	G	0.0	1.0					
EAC_SIGNED_R11				R	0.0	0.0	1.0					
EAC_SIGNED_RG11				R	G	0.0	1.0					
ETC2_SRGB8				R	G	B	1.0					
R16G16B16_UINT			DX10/OGL	R	G	B	1.0					
R16G16B16_SINT			DX10/OGL	R	G	B	1.0					
ETC2_RGB8_PTA				R	G	B	A					
ETC2_SRGB8_PTA				R	G	B	A					
ETC2_EAC_RGBA8				R	G	B	A					
ETC2_EAC_SRGB8_A8				R	G	B	A					
R8G8B8_UINT			DX10/OGL	R	G	B	1.0					
R8G8B8_SINT			DX10/OGL	R	G	B	1.0					
ASTC				R	G	B	A					
L8A8_UINT			DX10/OGL	L	L	L	A					
L8A8_SINT			DX10/OGL	L	L	L	A					
L8_UINT			DX10/OGL	L	L	L	1.0					
L8_SINT			DX10/OGL	L	L	L	1.0					
I8_UINT			DX10/OGL	I	I	I	I					
I8_SINT			DX10/OGL	I	I	I	I					

Programming Note	
Context:	SURFACE_STATE
Any surface format not supported by the 3D sampler except Raw Buffer format will return undefined results. The Unsupported Raw Buffer format will be treated as RGBA8888 by default.	

Programming Note	
Context:	SURFACE_STATE
Surface formats PLANAR_420_8 and PLANAR_420_16 which require half-pitch chroma planes (e.g. YV12) cannot support fenced tiling.	

Programming Note	
Context:	SURFACE_STATE/Shader channel select
It is recommended, for performance reasons, to never use any format of the type L*A*, I* or A*. Instead use R* or RG* in combination with **Shader Channel Select** .	

Programming Note	
Context:	SURFACE_STATE/Shader channel select
The BC2_NORM, BC3_UNORM, BC5_UNORM, BC5_SNORM and BC7_UNORM surface types must only be used when the **Sampler L2 Bypass Mode Disable** field in the RENDER_SURFACE_STATE is set.	

Programming Note	
Context:	EAC_SIGNED_R11 and EAC_SIGNED_RG11
When sampling on a EAC_SIGNED_R11 or EAC_SIGNED_RG11 , the returned value may be clamped to -1.0009765625 instead of -1.0	

Programming Note	
Context:	Anisotropic Filtering with 128bpt and Gen9 Border Color Mode
Anisotropic Filtering of 128bpt surfaces, when Sampler State **Texture Border Color Mode** is set to DX9, is not supported	

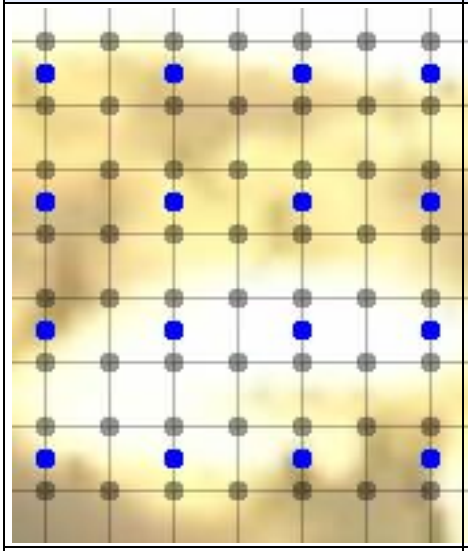
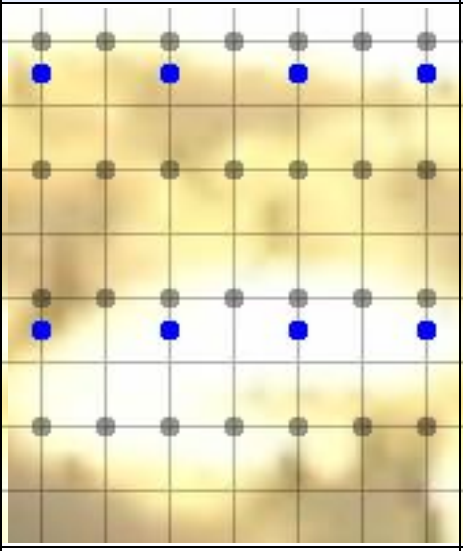
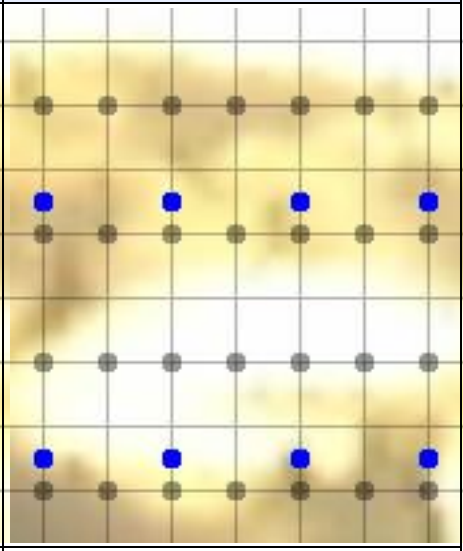
Programming Note	
Context:	NULL Surfaces and Shader Channel Select
Is SURFTYPE_NULL is selected, **Shader Channel Select Alpha** must be programmed to SCS_ZERO.	

SURFACE_STATE for Deinterlace sample_8x8 and VME

This section contains media surface state definitions.

Cr(V)/Cb(U) Pixel Offset V Direction

The position of Y is brown and the position of Cr(V)/Cb(U) is blue.

Full Frame	Top Field	Bottom Field
		
V Offset 0.5	V Offset 0.25	V Offset 0.75

MEDIA_SURFACE_STATE

Programming Note	
Context:	SURFACE_STATE for Deinterlace sample_8x
The Faulting modes described in the MEMORY_OBJECT_CONTROL_STATE should be set to the same for the multi-surface Video Analytics functions like “LBP Correlation” and “Correlation Search” for both the surfaces.	

SAMPLER_STATE

SAMPLER_STATE has different formats, depending on the message type used. For SNB+, the sample_8x8 and deinterlace messages use a different format of SAMPLER_STATE as detailed in the corresponding sections.

Restriction: The **Min LOD** and **Max LOD** fields need range increased from [0.0,13.0] to [0.0,14.0] and fractional bits increased from six to eight. This requires a few fields to be moved as indicated in the text.

Sampler State
SAMPLER STATE
SAMPLER_BORDER_COLOR_STATE

If border color is used, all formats must be provided. Hardware will choose the appropriate format based on **Surface Format** and **Texture Border Color Mode**. The values represented by each format should be

the same (other than being subject to range-based clamping and precision) to avoid unexpected behavior.

DWord	Bits	Description
0	31:24	**Border Color Alpha** Format = UNORM8
	23:16	**Border Color Blue** Format = UNORM8
	15:8	**Border Color Green** Format = UNORM8
	7:0	**Border Color Red** Format = UNORM8
1	31:0	**Border Color Red** Format = IEEE_FP
2	31:0	**Border Color Green** Format = IEEE_FP
3	31:0	**Border Color Blue** Format = IEEE_FP
4	31:0	**Border Color Alpha** Format = IEEE_FP
5	31:16	**Border Color Green** Format = FLOAT16
	15:0	**Border Color Red** Format = FLOAT16
6	31:16	**Border Color Alpha** Format = FLOAT16
	15:0	**Border Color Blue** Format = FLOAT16
7	31:16	**Border Color Green** Format = UNORM16
	15:0	**Border Color Red** Format = UNORM16
8	31:16	**Border Color Alpha** Format = UNORM16
	15:0	**Border Color Blue** Format = UNORM16
9	31:16	**Border Color Green** Format = SNORM16
	15:0	**Border Color Red** Format = SNORM16
10	31:16	**Border Color Alpha** Format = SNORM16

DWord	Bits	Description
	15:0	**Border Color Blue** Format = SNORM16
11	31:24	**Border Color Alpha** Format = SNORM8
	23:16	**Border Color Blue** Format = SNORM8
	15:8	**Border Color Green** Format = SNORM8
	7:0	**Border Color Red** Format = SNORM8

Border Color Programming for Integer Surface Formats

For integer formats, there are different possible cases depending on the bits per channel (bpc) and bits per texel (bpt) of the surface format.

Integer Surface Format – Different Types	Surface formats
32 bpc, 128 bpt case (4 types)	R32G32B32A32_UINT R32G32B32_UINT R32G32B32A32_SINT R32G32B32_SINT
16 bpc, 64 bpt case (5 types)	R16G16B16A16_UINT, R10G10B10A2_UINT X32_TYPELESS_G8X24_UINT R16G16B16_UINT R16G16B16A16_SINT R16G16B16_SINT
32 bpc, 64 bpt case (2 types)	R32G32_UINT R32G32_SINT
8 bpc, 32 bpt cases (9 types)	R8G8B8A8_UINT R8G8_UINT R8_UINT X24_TYPELESS_G8_UINT R8G8B8_UINT R8G8B8A8_SINT R8G8_SINT

Integer Surface Format – Different Types	Surface formats
	R8_SINT R8G8B8_SINT
16 bpc, 32 bpt cases (4 types)	R16G16_UINT R16_UINT R16G16_SINT R16_SINT
32 bpc, 32 bpt case (2 types)	R32_UINT R32_SINT

HW supports only 1 index for a given Sampler Border Color state and Sampler State. So, SW will have to program the table in **SAMPLER_BORDER_COLOR_STATE** at DWord offsets 16 to 19, as per the integer surface format type (depends on the bits per channel and bits per texel of the surface format). If any color channel is missing from the surface format, the corresponding border color should be programmed as zero; if the alpha channel is missing, the corresponding Alpha border color should be programmed as 1. Some of the representative cases are listed below:

Case 2: R32G32B32A32_SINT (32 bpc, 128 bpt, 4 channel, SINT)

Each of the values in the above table would have be to programmed as sint32 value.

Case 3: R32G32B32_UINT (32 bpc, 128 bpt, 3 channel)

R/G/B values would be programmed like in Case1. Alpha channel value at DWORDN+3 would have to be programmed as Integer 1.

Sampler State
3DSTATE_CHROMA_KEY
3DSTATE_SAMPLER_PALETTE_LOAD0
3DSTATE_MONOFILTER_SIZE
SAMPLER_INDIRECT_STATE

Messages

The sampler receives messages from shader clients. These messages contain information to allow the sampler to perform sample operations and return results. A message consists of four components:

- Execution Mask: Indicates, for a given SIMD, which pixels are valid.
- Message Descriptor: Required information including length of the message, and the lenght of the response
- Message Header: Optional information which may be required for certain operations (e.g. Direct Write to Render Target)
- Message Payload: Specific data for each sampler operation including coordinates and other message parameters.

Programming Notes

Programming Note	
Context:	3D Sampler Messages
Write back messages can return erroneous information in the Pixel Null Mask field for sample_c messages on a 64-bit surface format (e.g. R16G16B16A16_UNORM)	

Programming Note
A message header is required for GPGPU kernals in order to allow mid-thread pre-emption to allow save/restore mechanisms to work correctly.

Message Descriptor and Execution Mask

Execution Mask

SIMD16. The 16-bit execution mask forms the valid pixel signals. This determines which pixels are sampled and results returned to the GRF registers. Samples for invalid pixels are not overwritten in the GRF. However, if LOD needs to be computed for a subspan based on the message type and MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, as these are needed for the LOD computation.

SIMD8. The low 8 bits of the execution mask form the valid pixel signals. If LOD needs to be computed based on MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, since these are needed for the LOD computation.

SIMD4x2. The low 8 bits of the execution mask is interpreted in groups of four. If any of the high 4 bits are asserted, that sample is valid. If any of the low 4 bits are asserted, that sample is valid. The **Write Channel Mask** rather than the execution mask determines which channels are written back to the GRF.

SIMD32. The execution mask is ignored, all pixels are considered valid, and all channels are returned regardless of the execution mask.

Message Descriptor

The Sampler Message Descriptor is composed of a 32-bit Message Descriptor and a 32-bit Extended Message Descriptor (used for support Bindless resources, etc.). The lower 11 bits of the Extended Message Descriptor are defined by the Extended Message Descriptor for the Execution Unit.

The message descriptor is sent in parallel with the message payload (and header if used).

Descriptor
Message Descriptor - Sampling Engine
Extended Message Descriptor - Sampling Engine
Extended Message Descriptor - Execution Unit

Restrictions for Message Descriptors

Programming Note	
Context:	3D Sampler Messages
Use of any message to the Sampling Engine function with the **End of Thread** bit set in the message descriptor is not allowed.	

Message Header

The message header for the sampling engine is the same regardless of the message type. The messasge header is optional. If the header is not present, the behavior is as if the message was sent with all fields in the header set to zero and the write channel masks are all enabled and offsets are zero. However, if the header is not included in the message, the Sampler State Pointer will be obtained from the command stream input for the given thread.

When Response length is 0 for sample_8x8 message then the data from sampler is directly written out to memory using media write message.

DWord	Bits	Description
M0.5	31:5	Reserved
M0.5	4:0	Output format. Only for Sample_8x8 message with direct HDC write: 0 YCRCB_NORMAL 1 YCRCB_SWAPUVY 2 YCRCB_SWAPUV 3 YCRCB_SWAPY 4 PLANAR_420_8 (NV12 only) 5 Y8_UNORM 6 Y16_SNORM 7-16 Reserved 17 Y32_UNORM 18 Reserved
M0.4	31:16	Destination Y Address (u16)

DWord	Bits	Description
M0.4	15:0	Destination X Address in bytes (u16) X Address should always be DWord-aligned.
M0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the sampler state table. This field is ignored for *ld* and *resinfo* message types. This pointer is relative to the **Dynamic State Base Address**. Format = DynamicStateOfffset[31:5] The Sampler State Pointer does not have to be defined by the Message Header (many messages do not require a message header). The Sampler State Pointer may be delivered from the Command Streamer without the need for a Message Header.
M0.3	4:0	Ignored
M0.3	31:4	**Sampler State Pointer:** Specifies the 16-byte aligned pointer to the sampler state table. This field is ignored for *ld* and *resinfo* message types. This pointer is relative to the **Dynamic State Base Address** or **Bindless Surface State Base Address** depending on the setting of Bindless Surface Base Address Select bit in the GT_Mode Register. Format = StateOffset[31:4] The Sampler State Pointer does not have to be defined by the Message Header (many messages do not require a message header). The Sampler State Pointer may be delivered from the Command Streamer without the need for a Message Header.
M0.2	31:24	**Render Target or Destination Binding Table Index** Specifies the index into the binding table for the render target or HDC for messages with response length of zero (the binding table index for the sampler surface is in the message descriptor). Format = U8 Range = [0,255]
M0.2	23	**Pixel Null Mask Enable** Specifies whether the writeback message includes an extra phase indicating the pixel null mask. Refer to the *Writeback Message* section for details on format. This field must be disabled for sample+killpix and all SIMD32/64 messages. Format = Enable Ignored for Sample_8x8 message

DWord	Bits	Description
M0.2	22	**SIMD Mode Extension** If SIMD Mode in the message descriptor is set to SIMD8D/SIMD4x2, this field specifies which mode is used. For other SIMD Modes, this field is ignored. 0: SIMD8D 1: SIMD4x2
M0.2	21	Reserved
M0.2	20	Reserved
M0.2	19:18	**SIMD32/64 Output Format Control** Specifies the output format of SIMD32/64 messages (sample_unorm* and sample_8x8). Ignored for other message types. Refer to the writeback message formats for details on how this field affects returned data. This field is ignored for sample_8x8 messages if the Function is not AVS and MinMaxFilter. For MinMaxFilter only 16 bit Full and 8 bit Full modes are supported. This field is ignored and not used for HDC write message. 0: 16 bit Full 1: 16 bit Chrominance Downsampled 2: 8 bit Full 3: 8 bit Chrominance Downsampled This feature should be programmed to 0h because non-0 values may cause data corruption in returned values.
M0.2	17:16	**Gather4 Source Channel Select:** Selects the source channel to be sampled in the gather4* messages. Ignored for other message types. 0: Red channel 1: Green channel 2: Blue channel 3: Alpha channel For gather4*_c messages, this field must be set to 0 (Red channel).

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>M0.2</td><td>15</td><td>Alpha Write Channel Mask: Enables the alpha channel to be written back to the originating thread.
0: Alpha channel is written back.
1: Alpha channel is not written back.
<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>3D Sampler Messages</td></tr><tr><td colspan="2">A message with all four channels masked is not allowed.
This field is ignored for the deinterlace message.
This field must be set to zero for sample_8x8 in VSA mode.
For Sample_8x8 messages, Alpha/Blue/Red channels should be always masked (set to 1) and only Green channel is enabled (set to 0).
This field must be set to zero for all gather4* messages.</td></tr></table></td></tr>
<tr><td>M0.2</td><td>14</td><td>Blue Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>13</td><td>Green Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>12</td><td>Red Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>11.8</td><td>U Offset: The u offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement.
<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>3D Sampler Messages</td></tr><tr><td colspan="2">This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
This field is ignored if the offu parameter is included in the gather4* messages.</td></tr></table>
<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>Non-Normalized Floating-Point Coordinates</td></tr><tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr></table></td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>M0.2</td><td>7:4</td><td>V Offset: The v offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>3DSampler Messages</td></tr><tr><td colspan="2">This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
This field is ignored if the offu parameter is included in the gather4* messages.</td></tr></table>
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>Non-Normalized Floating-Point Coordinates</td></tr><tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr></table></td></tr>
<tr><td>M0.2</td><td>3:0</td><td>R Offset: The r offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>3D Sampler Messages</td></tr><tr><td colspan="2">This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.</td></tr></table>
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>Non-Normalized Floating-Point Coordinates</td></tr><tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr></table></td></tr>
<tr><td>M0.1</td><td>31:0</td><td>Reserved</td></tr>
<tr><td>M0.0</td><td>31:0</td><td>Reserved</td></tr>
</table>

Message Types

3D Sampler Message Types

The 3D sampler supports multiple message types with different types of behaviors being supported. Each message can be supported with multiple SIMD forms (e.g. SIMD8, SIMD16 etc). See the section Message Formats for which SIMD forms are supported as well as the specific parameters and order of parameters for each message.

Below is a complete list of supported 3D Sampler message types:

Message Types
sample_*
ld_*
gather4_*
LOD
sampleinfo
resinfo
cache_flush
sample_unorm1
sample_8x81

[1] For SIMD32/SIMD64 Media-type messages such as sample_unorm, sample_8X8 and deinterlace see the Media Sampler SIMD32/64 section for a description of the payloads of these messages and the expected behavior.

Common Message Variants

Many message types have multiple variants which provide for different sampling behaviors. The variants of a message type are named by appending a suffix to the base message.

Variant Suffix	Definition
_l	LOD Override: The LOD is provided in the message rather than being calculated from the gradients of the sub-span pixel coordinates.
_b	LOD Bias: A floating-point value between +16.0 and -16.0 is added to the LOD based on gradients of the sub-span pixel coordinates.
_c	Compare: Returns a white or black result depending on the comparison of a Ref parameter to the resulting red-channel of the sample. Comparison type is defined by the **Shadow Function** field in the SAMPLER_STATE.
_lz	LOD=0 Override: LOD is forced to 0. No LOD is calculated or provided in the message.
_d	Gradient: Rather than receiving absolute texel coordinates for all pixels of sub-span, a single pixel coordinate tuple is provided and a set of floating-point gradient values with respect to those pixel

Variant Suffix	Definition
	coordinates. This is then used to calculate other pixel coodinates and the LOD
_po	Pixel Offset: Used only with gather4-type messages, it means the message provides a set of non-normalized integer texel offsets between +31 and -32 which are added to the calculated surface position.
_killpix	Kill Pixel: Used in conjunction with the Chroma Key mode enabled by the Sampler State Field CHROMA_KEY_ENABLE. It causes the associated sampler result to include a "Kill Pixel Mask" where 0's indicate pixels which matched a particular Chroma Key Mode.
_min	Minimum Value: Forces return of the minimum channel value for all contributing texels a filter operation.
_max	Maximum Value: Forces return of the maximum channel value for all contributing texels a filter operation.

See the subsections for each message type for a description of the behaviors and programming restrictions.

Restrictions and Programming Notes for All Message Types

Programming Note	
Context:	Message Types
For surfaces of type SURFTYPE_CUBE, the sampling engine requires u, v, and r parameters that have already been divided by the absolute value of the parameter (u, v, or r) with the largest absolute value.	

Programming Note	
Context:	Reduction Filter and *_c message variants.
Programming restrictions defined for *_c variants of all message types will not apply if the reduction filter is enabled via the Reduction Type Enable field in the SAMPLER_STATE and a Reduction Type that is **NOT** COMPARISON is selected. The restrictions of the non *_c variant will apply instead.	

Sample Message Types

Sample Message Definition

Message Type	Description
sample_*	The surface is sampled using the indicated sampler state. LOD is computed using calculated gradients between adjacent pixels. One, two, or three floating-point coordinate parameters may be specified depending on how many dimensions the indicated surface type uses. Extra parameters specified are ignored. Missing parameters are defaulted to 0. The sample message enables filtering/blending operations (e.g. MAP_LINEAR) and addressing modes

Message Type	Description
	(e.g. Mirror) which are controlled by SAMPLER_STATE.

Supported Variants

Message	Description
sample	Basic sample operation as described above.
sample_l	Basic sample with LOD in message, not computed
sample_b	Basic sample with Bias in messaged added to computed LOD.
sample_c	Basic sample with Reference value in message and comparison against Red channel of the sampled surface returned. Used primarily for shadow maps. Comparison type is defined by the **Shadow Function** field in the SAMPLER_STATE.
sample_lz	Basic sample with LOD forced to 0. Possibly higher performance than **sample_l** for cases where the surface has MIP_COUNT=1 because the LOD does not need to be computed or sent in message.
sample_l_c	sample_c with LOD override in message, not computed.
sample_b_c	sample_c with LOD Bias
sample_c_lz	Similar to sample_c with LOD forced to 0. Possibly higher performance than **sample_c** or **sample_l_c** for cases where the surface has MIP_COUNT=1 because the LOD does not need to be computed.
sample_d	The surface is sampled using the indicated sampler state. LOD is computed using the gradients present in the message. The *r* coordinate and its gradients are required only for surface types that use the third coordinate. Usage of this message type on cube surfaces assumes that the u, v, and gradients have already been transformed onto the appropriate face, but still in [-1,+1] range. The *r* coordinate contains the faceid, and the r gradients are ignored by hardware. Previously known as sample_g.
sample_d_c	Same as sample_d, but returns comparison against Red Channel of sampled surface like sample_c. Previously known as sample_g_c.
sample_killpix	Basic sample, but returns a Kill Pixel Mask based on Chroma Key sampler comparison results. An additional register is returned after the sample results which contains the kill pixel mask. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask.
sample_min	Basic sample, but returns minimum value of all contributing texels for each individual color channel.
sample_max	Basic sample, but returns maximum value of all contributing texels for each individual color channel.

Restrictions and Programming Notes for Sample

Programming Note
sample: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE.

Programming Note
The Surface Format of the associated surface cannot be MONO8. Sample is not supported in SIMD4x2 mode.
sample: Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
sample: The *mlod* parameter specifies the per-pixel MinLOD.

Restrictions and Programming Notes for sample_b

Programming Note
sample_b: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. The LOD bias delivered in the **bias** parameter is restricted to a range of [-16.0, +16.0). Values outside this range produce undefined results.
sample_b is not supported in SIMD4x2 mode.
sample_b: The *mlod* parameter specifies the per-pixel MinLOD.

Restrictions and Programming Notes for sample_l and sample_lz

Programming Note
sample_l and sample_lz: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
sample_l and sample_lz: The ld_lz message is the same as ld except the lod parameter is set to zero instead of being delivered.

Restrictions and Programming Notes for sample_c and sample_c_lz

Programming Note
sample_c and sample_c_lz: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE. The Surface Format of the associated surface must be indicated as supporting shadow mapping as indicated in the surface format table. MIPFILTER_LINEAR, MAPFILTER_LINEAR, MAPFILTER_ANISOTROPIC are allowed even for surface formats that are listed as not supporting filtering in the surface formats table. Use of the SIMD4x2 form of *sample_c* without **Force LOD to Zero** enabled in the message header is not allowed, as it is not possible for the hardware to compute LOD for SIMD4x2 messages.

Programming Note
Using SURFTYPE_CUBE surfaces is undefined with the following surface formats: I24X8_UNORM, L24X8_UNORM, A24X8_UNORM, I32_FLOAT, L32_FLOAT, and A32_FLOAT. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. Using DX9 **Texture Border Color Mode** and either of the following is undefined: • Any applicable Address Control Mode (depending on Surface Type) is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER. • Surface Type is SURFTYPE_CUBE and any Cube Face Enable is disabled.
sample_c is not supported in SIMD4x2 mode.
sample_c and sample_c_lz: Must use the DX10 BORDER_COLOR_MODE (selected in sampler state) when using CLAMP_BORDER addressing mode.
The *mlod* parameter specifies the per-pixel MinLOD.

Restrictions and Programming Notes for sample_l_c

Programming Note
sample_l_c: All restrictions applying to both sample_l and sample_c must be honored.
sample_l_c is allowed as a SIMD4x2 message.
sample_l_c: Must use the DX10 BORDER_COLOR_MODE (selected in sampler state) when using CLAMP_BORDER addressing mode.

Restrictions and Programming Notes for sample_b_c

Programming Note
sample_b_c: All restrictions applying to both sample_bl and sample_c must be honored.
sample_b_c: All variations of the sample_c, must use the DX10 BORDER_COLOR_MODE (selected in sampler state) when using CLAMP_BORDER addressing mode.

Restrictions and Programming Notes for sample_d

Programming Note
sample_d: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
sample_d: The mlod parameter specifies the per-pixel MinLOD.

Restrictions and Programming Notes for sample_d_c

Programming Note
sample_d_c: All restrictions applying to both sample_d and sample_c must be honored.
sample_d_c is allowed as a SIMD4x2 message.

Restrictions and Programming Notes for sample_killpix

Programming Note
sample_killpix: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. The Surface Format of the associated surface cannot be MONO8. sample_killpix is supported only in SIMD8 mode. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
sample_killpix: If the sampler is disable via the **Sampler Disable** bit in the SAMPLER_STATE, the Kill Pixel Mask returned will be undefined.

Restrictions and Programming Notes for sample_min, sample_max

Programming Note
The Surface Type of the associated surface must be SURFTYPE_2D and **Surface Array** must be disabled. The Surface Format of the associated surface cannot be MONO8 Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1
If sampling on a 3D surface (Volumetric),the following two conditions cannot be supported on a sample_min or sample_max at the same time: • Sampler state addresse mode for q direction specifies D3D11_TEXADDRESS_BORDER • Address mode for U and V do not specify D3D11_TEXADDRESS_BORDER

ld Message Types

ld Message Definition

Supported Variants

Message	Description
ld	Basic ld operation as described above.
ld_lz	Basic ld with LOD forced to 0. Reduces the size of the message with potential performance and power advantage becasuse LOD is not sent as part of the message.

Message	Description
ld2dms	Basic ld operation but to a multi-sampled surface. Includes additional parameters for MCS values (see ld_mcs below) and Si (sample index) for each pixel in the message. Can be used for all multi-sampled surfaces up to X8 MSAA amd for X16 MSAA in cases where the upper half of the sub-samples (planes 8 thru 15) all reference plane0 (see ld_mcs to see how determining X16 MSAA viability is determined).
ld2dms_w	Same as ld2dms but provides more bits for MCS to support multi-sampled surfaces which are X16 MSAA. Therefore, the ld2dms_w message has two MCS parameters, named mcsl and mcsh. These parameters contain the low and high halves of the 64-bit MCS value, respectively. However, use of the ld2dms_w message is allowed with Number of Multisamples set to 2, 4, 8, or 16. With this message, if Number of Multisamples is 2, 4 or 8, the ld2dms_w message behaves the same as ld2dms, with mcsl used for mcs, and mcsh ignored. For SIMD8/SIMD16 the mcsh parameters is only used for X16 MSAA, and should be all 0's for X8 MSAA or lower depth.
ld2dss	Same as ld2dms, but does not require MCS, and uses only ssi parameter to pick same sub-slice for all pixels. The ssi parameter defines the sample slice that will be sampled from. Refer to the multisample storage format in the GPU Overview volume for more details.
ld_mcs	Basic ld operation, but used to fetch the MCS auxiliary surface data rather than pixels from the surface itself. The returned MCS data is used to construct the ld2dms/ld2dms_w/ld2dss message to sample from the multi-sampled surface.

Restrictions and Programming Notes for ld

Programming Note
ld: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_BUFFER for the ld message. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1

Restrictions and Programming Notes for ld_lz

Programming Note
ld_lz: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_BUFFER for the ld message. The Surface Format of the associated surface cannot be MONO8.

Restrictions and Programming Notes for ld_mcs

Programming Note

Programming Note
ld_mcs: The Surface Type of the surface associated with the auxiliary surface must be SURFTYPE_2D.
ld_mcs: The ld_mcs message uses the Auxiliary Surface Base Address and Auxiliary Surface Pitch fields in SURFACE_STATE to determine the base address and pitch of the surface. Surface Format is overridden to R8_UINT if Number of Multisamples is 2 or 4, R32_UINT if Number of Multisamples is 8, or R32G32_UINT if Number of Multisamples is 16. This message cannot be used on a non-multisampled surface. Otherwise, ld_mcs behaves like the ld message. If ld_mcs is issued on a surface with Auxiliary Surface Mode not set to AUX_MCS, this message returns zeros in all channels.
ld_mcs: If ld_mcs is issued on a surface with Auxiliary Surface Mode not set to AUX_CCS_D, this message returns zeros in all channels. Ld_mcs always returns 512 bits of data for SIMD8 and 1024 bits of data for SIMD16 regardless of the MSAA depth of the surface. If ld_mcs is issued on an auxiliary surface associated with a X8 MSAA or lower depth, the upper-half of the return data will be 0's.
ld_mcs: All LOD parameters must be the same value within a single message (i.e. all samples retured for message must be from a single MIP).

Restrictions and Programming Notes for ld2dms

Programming Note
ld2dms: The sampled surface type must not be MULTISAMPLECOUNT_1 The sample surface type must not be MULTISAMPLECOUNT_16 any of the upper 8 planes contain valie data. The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface cannot be MONO8.
ld_2dms: All LOD parameters must be the same value within a single message (i.e. all samples retured for message must be from a single MIP).

Restrictions and Programming Notes for ld_2dms_w

Programming Note
ld_2dms_w: The Surface type must not be MULTISAMPLECOUNT_1.

Programming Note
The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface cannot be MONO8.
ld_2dms: All LOD parameters must be the same value within a single message (i.e. all samples retured for message must be from a single MIP).

Restrictions and Programming Notes for ld_2dss_w

Programming Note
ld_2dss: All LOD parameters must be the same value within a single message (i.e. all samples retured for message must be from a single MIP).

gather4 Message Types

Definitions

Message Type	Description
gather4_*	The surface is sampled using MAP_LINEAR filtering, regardless of the filtering mode specified in the sampler state and with LOD forced to zero of the visible resource. However, the samples are not filtered. Instead, the selected color channel the four contributing texels are returned directly in the sample's corresponding four channels as follows: upper left sample = alpha channel / upper right sample = blue channel lower left sample = red channel / lower right sample = green channel Returned color channel is selected through a message field: **Gather4 Source Channel Select** Two or three floating-point coordinates may be specified depending on how many coordinate dimensions the indicated surface type uses. Extra parameters specified are ignored. Missing parameters default to 0.

Supported Variants:

Message	Description
gather4	Basic gather4 behavior as described above.
gather4_c	Similar to basic gather4 but performs a compare between a Reference parameters and the gathered

Message	Description
	pixels and returns a white or black (1 or 0). In addition, like **sample_c**, it only returns data on the Red channel, so the **Gather4 Source Channel Select** must be set to Red.
gather4_po	Similar to gather4, but includes offu and offv parameters, which contain texel-space offsets that override the U/V Offset fields in the message header. Unlike the message header fields however, these offsets have a wider range [-32,+31], and can differ per pixel or sample. The format of the data is 32-bit 2's complement signed integer, but hardware only interprets the least significant 6 bits of each value, treating it as a 6-bit 2's complement signed integer
gather4_po_c	This is a combination of gather4_c and gather4_po, and will return data only on the Red Channel (**Gather4 Source Channel Select** must be set to Red), but uses offu and offv to shift the position on the surface.

Restrictions and Programming Notes for gather4:

Programming Note
gather4: The Surface Type of the associated surface must be SURFTYPE_2D or SURFTYPE_CUBE. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. Using DX9 Border Color Mode when either of the following is true will yield undefined results: • Any applicable Address Control Mode (depending on Surface Type) is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER. • Surface Type is SURFTYPE_CUBE and any Cube Face Enable is disabled.
gather4: If Surface Format is R10G10B10_SNORM_A2_UNORM and Gather4 Source Channel Select is alpha channel, the returned value may be incorrect.
gather4: When using gather4 type messages on CUBEMAP surfaces with SINT* surface formats the ChromaKeyEnable state bit should be set and the ChromaKeyMode state set to KEYFILTER_KILL_ON_ANY_MATCH.

Restrictions and Programming Notes for gather4_c:

Programming Note
gather4_c: The Surface Type of the associated surface must be SURFTYPE_2D or SURFTYPE_CUBE. The Surface Format of the associated surface must be one of the following: R32_FLOAT_X8X24_TYPELESS, R32_FLOAT, R24_UNORM_X8_TYPELESS, or R16_UNORM. The channel selected by the Gather4 Source Channel Select field in the message header must be set to **Red**. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. Using DX9 Border Color Mode and either of the following is undefined:

Programming Note
• Any applicable Address Control Mode (depending on Surface Type) is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER. • Surface Type is SURFTYPE_CUBE and any Cube Face Enable is disabled.
gather4_c: When using gather4 type messages on CUBEMAP surfaces with SINT* surface formats the ChromaKeyEnable state bit should be set and the ChromaKeyMode state set to KEYFILTER_KILL_ON_ANY_MATCH.

Restrictions and Programming Notes for gather4_po:

Programming Note
gather4_po: The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. Using DX9 Border Color Mode and either of the following is undefined: • Any applicable Address Control Mode (depending on Surface Type) is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER. Surface Type is SURFTYPE_CUBE and any Cube Face Enable is disabled.
gather4_po: When using gather4 type messages on CUBEMAP surfaces with SINT* surface formats the ChromaKeyEnable state bit should be set and the ChromaKeyMode state set to KEYFILTER_KILL_ON_ANY_MATCH.

Restrictions and Programming Notes for gather4_po_c:

Programming Note
gather4_po_c: The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface must be one of the following: R32_FLOAT_X8X24_TYPELESS, R32_FLOAT, R24_UNORM_X8_TYPELESS, or R16_UNORM The channel selected by the Gather4 Source Channel Select field in the message header must be set to **Red**. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1. Using DX9 Border Color Mode and either of the following is undefined: • Any applicable Address Control Mode (depending on Surface Type) is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER. • Surface Type is SURFTYPE_CUBE and any Cube Face Enable is disabled.
gather4_po_c: When using gather4 type messages on CUBEMAP surfaces with SINT* surface formats the ChromaKeyEnable state bit should be set and the ChromaKeyMode state set to KEYFILTER_KILL_ON_ANY_MATCH.

sampleinfo Message Type

sampleinfo Message Definition

Message Type	Description
sampleinfo	The surface indicated in the surface state is not sampled. Instead, the number of samples (UINT32) and the sample position palette index (UINT32) for the surface are returned in the red and alpha channels respectively as UINT32 values. The sample position palette index returned in alpha is incremented by one from its value in the surface state. The **Sampler State Pointer** and **Sampler Index** are ignored. The Surface Type of the associated surface must be SURFTYPE_2D or SURFTYPE_NULL.

Supported Variants:

None

Restrictions and Programming Notes for sampleinfo:

Programming Note
sampleinfo: The return format for sampleinfo message must be 32-bits.

LOD Message Type

LOD Message Definition

Message Type	Description
LOD	The surface indicated in the surface state is not sampled. Instead, LOD is computed as if the surface will be sampled, using the indicated sampler state, and the clamped and unclamped LOD values are returned in the red and green channels, respectively, in FLOAT32 format. The blue and alpha channels are undefined, and can be masked to avoid returning them. LOD is computed using gradients between adjacent pixels. Three parameters are always specified, with extra parameters not needed for the surface being ignored.

Supported Variants:

None

Restrictions and Programming Notes for LOD:

Programming Note
LOD: The Surface Type of the associated surface must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, or SURFTYPE_CUBE. The Surface Format of the associated surface cannot be MONO8. Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
LOD is not supported in SIMD4x2 mode.
LOD: The return format for LOD message must be 32-bits.

resinfo Message Type

resinfo Message Definition

Message Type	Description
resinfo	The surface indicated in the surface state is not sampled. Instead, the width, height, depth, and MIP count of the surface are returned as indicated in the table below. The format of the returned data is UINT32. In the case of a 1D or 2D surfaces, the depth value returned is the number of array slices. For non-arrayed 1D and 2D surfaces the value returned will be 1. The width, height, and depth may be scaled by the LOD parameter provided in the message to give the correct dimensions of the specified mip level. The LOD parameter is an unsigned 32-bit integer in this mode (note that negative values are out-of-range when interpreted as unsigned integers). The Sampler State Pointer and Sampler Index are always ignored. For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, if the delivered LOD is outside of the range [0..MipCount-1], the returned values in the red, green, and blue channels are 0s.

Returned Parameters for **resinfo:**

For the resinfo message the Red, Green and Blue channels returned contain the following specialized information.

Surface Type	Red	Green	Blue	Alpha
SURFTYPE_1D	(Width+1)»LOD	Surface Array ? Depth+1 : 0	0	MIP Count
SURFTYPE_2D	(Width+1)»LOD) * (QuiltWidth+1)	((Height+1)»LOD)*	Surface Array ? Depth+1 : 0	MIP Count

Surface Type	Red	Green	Blue	Alpha
		(QuiltHeight+1)		
SURFTYPE_3D	(Width+1)»LOD	(Height+1)»LOD	(Depth+1)»LOD	MIP Count
SURFTYPE_CUBE	(Width+1)»LOD	(Height+1)»LOD	Surface Array ? Depth+1 : 0	MIP Count
SURFTYPE_BUFFER SURFTYPE_STRBUF	Buffer size (from combined Depth/Height/Width) If buffer size is exactly 2^32, zero is returned in this field.	Undefined	Undefined	Undefined
SURFTYPE_NULL	0	0	0	0

Supported Variants:

None

Restrictions and Programming Notes for resinfo:

Programming Note
resinfo: When a surface of type SURFTYPE_NULL is accessed by resinfo, the MIPCount returned is undefined instead of 0.
resinfo: The return format of resinfo message must be 32-bits.

cache_flush Message Type

cache_flush Message Definition

Message Type	Description
cache_flush	The texture caches in the sampling engine are invalidated. This includes all levels of texture cache, however the state caches are not invalidated. Any outstanding sample operations which arrive at the texture sampler prior to the cache_flush message are completed normally. Any sample operations which arrive after the cache_flush message will cause texture cache-misses and sampler will fetch textures from memory.

Supported Variants:

None

Restrictions and Programming Notes for cache_flush:

Programming Note
cache_flush: Software must ensure that this message is ordered appropriately with other messages. Hardware ensures only that this message is processed in the same order as the messages are received. The message causes a write back message to indicate that the flush has been completed. This message must be sent as a SIMD32/64 type message with header only (no additional message contents)

Media Message Types

The 3D sampler supports specific message types which are intended for use by the Media sampler. These messages include:

Message
sample_unorm
sample_8X8

Media message types are described in more detail, with payload information defined in the Media Sampler Section.

sample_unorm Message Types

sample_unorm Message Definition

Message Type	Description
sample_unorm_*	The surface is sampled using the indicated sampler state. 32 contiguous pixels in a 8-wide by 4-high block are sampled. The U and V addresses for the upper left pixel are delivered in this message along with Delta U and Delta V floating-point parameters and . Given a pixel at (x,y) relative to the upper left pixel (where (0,0) is the upper left pixel), the U and V for that pixel are computed as follows: U(x,y) = U(0,0) + DeltaU * x V(x,y) = V(0,0) + DeltaV * y Typically this message and its variants are used for media applications.

Supported Variants:

Message	Description
sample_unorm	Basic sample_unorm behavior as described above.
sample_unorm_RG	Similar to sample_unorm, but only the Red and Green Channels are returned because there is an implied write mask.
sample_unorm_killpix	This message is identical to the sample_unorm message except it returns a kill pixel

Message	Description
	mask in addition to the selected channels in the writeback message. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask. All restrictions of the sample_unorm message apply to this message also.
sample_unorm_RG_killpix	This message is identical to the sample_unorm_RG message except it returns a kill pixel mask in addition to the selected channels in the writeback message. This message type is required to allow the result of a chroma key enabled sampler in KEYFILTER_KILL_ON_ANY_MATCH mode to affect the final pixel mask. All restrictions of the sample_unorm message apply to this message also.

Restrictions and Programming Notes for sample_unorm, sample_unorm_RG, sample_unorm_killpix, sample_unorm_RG_killpix:

Programming Note
The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface must be UNORM with <= 8 bits per channel. The MIP Count, Depth, Surface Min LOD, Resource Min LOD, and Min Array Element of the associated surface must be 0. The Min and Mag Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR. The Mip Mode Filter must be MIPFILTER_NONE. The TCX and TCY Address Control Mode cannot be any of: • TEXCOORDMODE_CLAMP_BORDER • TEXCOORDMODE_HALF_BORDER • TEXCOORDMODE_MIRROR_ONCE • TEXCOORDMODE_WRAP Map Width must be >= 4. Map Height must be >= 4 • Number of Multisamples on the associated surface must be MULTISAMPLECOUNT_1.
Map Width must be >=16 Map Height must be >=16

sample_8x8 Message Type

Supported Variants:

None.

Restrictions and Programming Notes for sample_8x8:

Programming Note
The Surface Type of the associated surface must be SURFTYPE_2D. The Surface Format of the associated surface must be UNORM with <= 10 bits per channel. *DeltaV * Height* of the associated surface must be less than 16.0. • *Map Width* must be >= 4.
Supports functionality such as convolve, MinMax, MinMaxFilter, Dilate, Erode, BoolCentroid, and Centroid have been added on top of the existing sample_8x8. The convolve, MinMax, and MinMaxFilter are 16-wide X 4-high; the rest have variable size depending on the message.. See Media Sampler section for a description of how these modes work.

Message Format

The following section describes how messages are formatted, including how message length and response length are calculated. The sampler supports a variety of dfferent message types which perform different sampling and filtering operations (see Message Types section for a description of these sampler and filtering operations). For each message type, different SIMD modes are supported which determines how many pixels the sampler performs the specified operation on in parallel. The precision of the input parameters and resulting samples can also be controlled by programming the **Return Format** bit in the message descriptor (for sample precision) and using SIMD8H/SIMD8 and SIMD16H/SIMD16 message types (for coordinate precision).

Supported SIMD Types

The table below describes the SIMD modes which are supported. SIMD32 and SIMD64 are used for media-type operations only.

List of Supported Message SIMD Types

SIMD Modes	Description
SIMD4X2	
SIMD8	8 samples with 32-bit Coordinates
SIMD16	16 samples with 32-bit coordinates
SIMD32/64	Media Operations on 32 or 64 pixels
SIMD8D	8 pixels in 2 sub-spans with 64-bit Coordinates

Message Length

The **SIMD Mode** field determines the number of instances (i.e. pixels) to be sampled and the formatting of messages. The **Message Length** field indicates the number of parameters sent with the message. Higher-numbered parameters are optional, and default to a value of 0 if not sent but needed for the surface being sampled. The **Header Present** field determines whether a header is delivered as the first phase of the message or the default header from R0 of the thread's dispatch is used. A header will increase the length of the message by 1. Note that messages with more than 4 parameters will perform lower due to the additional information being sent to the sampler than messages with 4 or fewer parameters. For example, a sample message using 5 parameters will not be able to sustain the same throughput as a sample message with only 4 valid parameters.

The table below shows all of the SIMD modes supported by the sampling engine and how the Message Length is calculated for each. The variable "N" is the number of valid parameters, and "H" is 1 if the header is present, 0 otherwise. The maximum message length allowed to the sampler is 11. The sample_d, sample_b_c, and sample_l_c messages are not allowed with a SIMD Mode of SIMD16 because the message length is greater than 11.

Message Lengths for Supported SIMD Modes

SIMD Mode	Message Length
SIMD4X2	H+(N/4)
SIMD8	H+N
SIMD8D	H+(2*N)
SIMD16	H+(2*N)

Programming Note	
Context:	3D Sampler Messages - Message Format
The Cache_Flush message is SIMD32, but has no payload, only a header. So, the length is 1 for Cache_Flush	

Response Length

The Response Length field determines the number of 256-bit registers which are used to receive the result of each message type for each SIMD mode. The value of k shown in the response length is 1 if killpix is enabled and 0 otherwise.

SIMD Mode	Return Precision (bits per channel)	Response Length
SIMD4X2	32	1
SIMD8	32	4+k
SIMD8	16	2**
SIMD8D	32	4+k
SIMD8D	16	2**

SIMD Mode	Return Precision (bits per channel)	Response Length
SIMD16	32	8*
SIMD16	16	4*

Notes for Determining Response Lengths Table

Symbol	Note
*	For SIMD16, phases in the response length are reduced by 2 for each channel that is masked. SIMD16 messages with six or more parameters exceed the maximum message length allowed, in which case they are not supported. This includes some forms of sample_b_c, sample_l_c, and gather4_po_c message types. Note that even for these messages, if 5 or fewer parameters are included in the message, the SIMD16 form of the message is allowed. SIMD16 forms of sample_d and sample_d_c are not allowed, regardless of the number of parameters sent.
**	For SIMD8*, phases in the response length are reduced by 1 for each channel that is masked.

SIMD32/SIMD64 Special Cases for Response Length

The SIMD32/SIMD64 modes have special response lengths depending on the message type.

Mnemonic	Payload Layout	Response Length
sample_unorm	Pixel Shader	8**
sample_unorm + killpix	Pixel Shader	9**
deinterlace	Pixel Shader	†
sample_unorm	Media	8**
sample_unorm + killpix	Media	9**
sample_8X8	Media	16*

** For sample_unorm, phases in the response length are reduced by 2 for each channel that is masked.

Sample8x8 Footnote

* For sample_8x8, phases in the response length are reduced by 4 for each channel that is masked.

† For deinterlace, response length depending on certain state fields. Refer to writeback message definition for details.

If **Header Present** is *disabled*, **Response Length** can be set to values in the table below for SIMD8 and SIMD16 messages other than *sample+killpix*. The setting of **Response Length** determines the **Write Channel Mask** fields that are used by the hardware according to the table below. **Response Length** values not indicated in the table are not valid.

SIMD Mode	Return Format	Response Length	Alpha Write Channel Mask	Blue Write Channel Mask	Green Write Channel Mask	Red Write Channel Mask
SIMD8*	32-bit/ 16-bit	1	1	1	1	0
	32-bit/ 16-bit	2	1	1	0	0
	32-bit/ 16-bit	3	1	0	0	0
	32-bit/ 16-bit	4	0	0	0	0
SIMD16*	32-bit	2	1	1	1	0
	16-bit	1	1	1	1	0
	32-bit	4	1	1	0	0
	16-bit	2	1	1	0	0
	32-bit	6	1	0	0	0
	16-bit	3	1	0	0	0
	16-bit	4	0	0	0	0
	32-bit	8	0	0	0	0

If **Pixel Fault Mask Enable** is enabled, response length must be set to a value one larger than that indicated in the tables above.

Programming Note	
Context:	3D Sampler Messages - Message Format
Parameter 0 is required.	

Message Formats

The table below describes the format of each message type for every supported SIMD type.

=The Message Type field in the message descriptor in combination with the Message Length to determine which message is being sent. The table defines all of the parameters sent for each message type. The position of the parameters in the payload is given in the section following corresponding to the *SIMD mode* given in the table. The Mnemonic column specifies the DX10-equivalent shader instructions expected to be translated to each message type.

Message parameters are typically formed in sequential 256-bit registers and sent in order to the sampler to form a complete message. All parameters are of type IEEE_Float, except those in the The ld*, resinfo, bufferinfo, and the offu, offv of the gather4_po[_c] instruction message types, which are of type signed integer. Any parameter indicated with a blank entry in the table is unused. A message register containing only unused parameters is not included as part of the message.

Programming Note	
Context:	3D Sampler Messages - Message Format
For output control modes other than "16 bit Full", the response lengths are reduced accordingly. Refer to the writeback message format definitions to determine the correct response length for each case.	

Parameter Types

sample*, LOD, and gather4 messages

For all of the sample*, LOD, and gather4 message types, all parameters are 32-bit floating point, except the 'mcs', 'offu', and 'offv' parameters. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Normalized values range from [0,1] across the surface, with values outside the surface behaving as specified by the **Address Control Mode** in that dimension. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension, with values outside the surface being clamped to the surface.

Programming Note	
Context:	3D Sampler Messages Parameter Types
For the SIMD8D type messages, the u* and v* parameters are normalized 64-bit floating point.	

Surface Type	u	v	r	ai
SURFTYPE1D	normalized ‘x’ coordinate	unnormalized array index	ignored	ignored
SURFTYPE_2D	normalized ‘x’ coordinate	normalized ‘y’ coordinate	unnormalized array index	ignored
SURFTYPE_3D	normalized ‘x’ coordinate	normalized ‘y’ coordinate	normalized ‘z’ coordinate	ignored
SURFTYPE_CUBE	normalized ‘x’ coordinate	normalized ‘y’ coordinate	normalized ‘z’ coordinate	unnormalized array index

mcs parameter

The ‘mcs’ parameter delivers the multisample control data. The format of this parameter is always a 32-bit unsigned integer. Refer to the section titled “Multisampled Surface Behavior” for details on this parameter.

Ld* messages

For the ld message types, all parameters are 32-bit unsigned integers, except the ‘mcs’ parameter. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension. Input of any value outside of the range returns zero.

Surface Type	u	v	r
SURFTYPE1D	unnormalized ‘x’ coordinate	unnormalized array index	ignored
SURFTYPE_2D	unnormalized ‘x’ coordinate	unnormalized ‘y’ coordinate	unnormalized array index
SURFTYPE_3D	unnormalized ‘x’ coordinate	unnormalized ‘y’ coordinate	unnormalized ‘z’ coordinate
SURFTYPE_BUFFER	unnormalized ‘x’ coordinate	ignored	ignored

SIMD Payloads

This section contains the SIMD payload definitions.

SIMD16 Payload

The payload of a SIMD16 message provides addresses for the sampling engine to process 16 entities (examples of an entity are vertex and pixel). The number of parameters required to sample the surface depends on the state that the sampler/surface is in. Each parameter takes two message registers, with 8 entities, each a 32-bit floating point value, being placed in each register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this. For example, a 2D map using "sample_b" needs only u, v, and bias, but must send the r parameter as well.

DWord	Bits	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2.7	31:0	**Subspan 3, Pixel 3 (lower right) Parameter 0**
M2.6	31:0	**Subspan 3, Pixel 2 (lower left) Parameter 0**
M2.5	31:0	**Subspan 3, Pixel 1 (upper right) Parameter 0**
M2.4	31:0	**Subspan 3, Pixel 0 (upper left) Parameter 0**

DWord	Bits	Description
M2.3	31:0	**Subspan 2, Pixel 3 (lower right) Parameter 0**
M2.2	31:0	**Subspan 2, Pixel 2 (lower left) Parameter 0**
M2.1	31:0	**Subspan 2, Pixel 1 (upper right) Parameter 0**
M2.0	31:0	**Subspan 2, Pixel 0 (upper left) Parameter 0**
M3 – Mn		Repeat packets 1 and 2 to cover all required parameters.

SIMD8 Payload

This message is intended to be used in a SIMD8 thread, or in pairs from a SIMD16 thread. Each message contains sample requests for just 8 pixels.

DWord	Bits	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2 – Mn		Repeat packet 1 to cover all required parameters.

SIMD8D Payload

This message is intended to be used in a SIMD8 thread, or in pairs from a SIMD16 thread. Each message contains sample requests for just 8 pixels. The "u" and "v" parameters are delivered in double precision floating point, and thus it takes two message phases to deliver 8 values. These are labeled in the Payload Parameter Definition table as "u0", "u1", "v0", and "v1". The number after the u/v indicates which subspan is contained in that parameter.

Parameters "u0", "u1", "v0", or "v1"

DWord	Bits	Description
Mn.7	31:0	Pixel 3 (lower right) Parameter n – upper 32 bits Specifies the value of the pixel's parameter n. The actual parameter that maps to parameter n is given in the table in the Payload Parameter Definition section. Format = Double precision IEEE Float, upper 32 bits
Mn.6	31:0	Pixel 3 (lower right) Parameter n – lower 32 bits Format = Double precision IEEE Float, lower 32 bits
Mn.5	31:0	Pixel 2 (lower left) Parameter n – upper 32 bits
Mn.4	31:0	Pixel 2 (lower left) Parameter n – lower 32 bits
Mn.3	31:0	Pixel 1 (upper right) Parameter n – upper 32 bits
Mn.2	31:0	Pixel 1 (upper right) Parameter n – lower 32 bits
Mn.1	31:0	Pixel 0 (upper left) Parameter n – upper 32 bits
Mn.0	31:0	Pixel 0 (upper left) Parameter n – lower 32 bits

All Other Parameters

DWord	Bits	Description
Mn.7	31:0	Subspan 1, Pixel 3 (lower right) Parameter n Specifies the value of the pixel's parameter n. The actual parameter that maps to parameter n is given in the table in the Payload Parameter Definition section. Format = IEEE Float
Mn.6	31:0	Subspan 1, Pixel 2 (lower left) Parameter n
Mn.5	31:0	Subspan 1, Pixel 1 (upper right) Parameter n
Mn.4	31:0	Subspan 1, Pixel 0 (upper left) Parameter n
Mn.3	31:0	Subspan 0, Pixel 3 (lower right) Parameter n
Mn.2	31:0	Subspan 0, Pixel 2 (lower left) Parameter n
Mn.1	31:0	Subspan 0, Pixel 1 (upper right) Parameter n
Mn.0	31:0	Subspan 0, Pixel 0 (upper left) Parameter n

SIMD4x2 Payload

DWord	Bits	Description
M1.7	31:0	**Sample 1 Parameter 3** Specifies the value of the pixel's parameter 3. The actual parameter that maps to parameter 3 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Sample 1 Parameter 2**
M1.5	31:0	**Sample 1 Parameter 1**
M1.4	31:0	**Sample 1 Parameter 0**
M1.3	31:0	**Sample 0 Parameter 3**
M1.2	31:0	**Sample 0 Parameter 2**
M1.1	31:0	**Sample 0 Parameter 1**
M1.0	31:0	**Sample 0 Parameter 0**
M2		Parameters 4-7 if present
M3		Parameters 8-11 if present

Writeback Message

Corresponding to the four input message definitions are four writeback messages. Each input message generates a corresponding writeback message of the same type .

Programming Note	
Context:	3D Sampler Writeback Message
The Pixel Null Mask field, when enabled via the **Pixel Null Mask Enable** will be incorect for sample_c when applied to a surface with 64-bit per texel format such as R16G16BA16_UNORM. Pixel Null mask Enable may incorrectly report pixels as referencing a Null surface.	

SIMD16

Return Format = 32-bit

A SIMD16 writeback message consists of up to 8 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the

writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3. The pixels written within each destination register is determined by the execution mask on the “send” instruction.

: If Pixel Null Mask Enable is enabled an additional register is included after the last channel register. Behavior of pixels that had a null texel contribution depends on the setting of **Null Pixel Behavior**.

DWord	Bit	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel’s red channel. Format = IEEE Float, S31 signed 2’s comp integer, or U32 unsigned integer.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1.7	31:0	**Subspan 3, Pixel 3 (lower right) Red**
W1.6	31:0	**Subspan 3, Pixel 2 (lower left) Red**
W1.5	31:0	**Subspan 3, Pixel 1 (upper right) Red**
W1.4	31:0	**Supspan 3, Pixel 0 (upper left) Red**
W1.3	31:0	**Subspan 2, Pixel 3 (lower right) Red**
W1.2	31:0	**Subspan 2, Pixel 2 (lower left) Red**
W1.1	31:0	**Subspan 2, Pixel 1 (upper right) Red**
W1.0	31:0	**Supspan 2, Pixel 0 (upper left) Red**
W2		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations

DWord	Bit	Description
W3		**Subspans 3 and 2 of Green:** See W1 definition for pixel locations
W4		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W5		**Subspans 3 and 2 of Blue:** See W1 definition for pixel locations
W6		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W7		**Subspans 3 and 2 of Alpha:** See W1 definition for pixel locations
W8.7:1		Reserved (not written): W8 is only delivered when **Pixel Fault Mask Enable** is enabled.
W8.0	31:16	Reserved: always written as 0xffff
W8.0	15:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

Return Format = 16-bit

:A SIMD16 writeback message with **Return Format** of 16-bit consists of up to 4 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the "send" instruction.

: If Pixel Null Mask Enable is enabled an additional register is included after the last channel register. Behavior of pixels that had a null texel contribution depends on the setting of **Null Pixel Behavior**.

DWord	Bit	Description
W0.7	31:16	**Subspan 3, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Half Float, S15 signed 2's comp integer, or U16 unsigned integer.
	15:0	**Subspan 3, Pixel 2 (lower left) Red**
W0.6	31:16	**Subspan 3, Pixel 1 (upper right) Red**
	15:0	**Supspan 3, Pixel 0 (upper left) Red**
W0.5	31:16	**Subspan 2, Pixel 3 (lower right) Red**
	15:0	**Subspan 2, Pixel 2 (lower left) Red**
W0.4	31:16	**Subspan 2, Pixel 1 (upper right) Red**
	15:0	**Supspan 2, Pixel 0 (upper left) Red**
W0.3	31:16	**Subspan 1, Pixel 3 (lower right) Red**

	15:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.2	31:16	**Subspan 1, Pixel 1 (upper right) Red**
	15:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.1	31:16	**Subspan 0, Pixel 3 (lower right) Red**
	15:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.0	31:16	**Subspan 0, Pixel 1 (upper right) Red**
	15:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Green:** See W0 definition for pixel locations
W2		**Blue:** See W0 definition for pixel locations
W3		**Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written): W4 is only delivered when **Pixel Fault Mask Enable** is enabled.
W4.0	31:16	Reserved: always written as 0xffff
W4.0	15:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

SIMD8

Return Format = 32-bit

: A SIMD8* writeback message consists of up to 4 destination registers (5 in the case of sample+killpix). Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the *send* instruction.

For the sample+killpix message types, an additional register (W4) is included after the last channel register.

: If **Pixel Null Mask Enable** is enabled an additional register is included after the last channel register. Behavior of pixels that had a null texel contribution depends on the setting of **Null Pixel Behavior**.

DWord	Bits	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**

DWord	Bits	Description
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W2		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W3		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written) **:** This W4 is only delivered for the sample+killpix message or when **Pixel Null Mask Enable** in Message Header is set.
W4.0	31:16	**Dispatch Pixel Mask:** For his field is always 0xffff to allow dword-based ANDing with the R0 header in the pixel shader thread for sample+killpix messages.
W4.0	15:0	**Active Pixel Mask:** For sample+killpix messages this field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode. Since the SIMD8 message applies to only 8 pixels, only the low 8 bits within this field are used. The high 8 bits are always set to 1.
W4.0	31:24	Reserved: always written as 0xff when **Pixel Null Mask Enable** in Message Header is set.
W4.0	23:16	Reseved: always written as 0xff when **Pixel Null Mask Enable** in Message Header is set.
W4.0	15:8	Reserved: always written as 0xff when **Pixel Null Mask Enable** in Message Header is set.
W4.0	7:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel when **Pixel Null Mask Enable** in Message Header is set.

Return Format = 16-bit

[CHV], :A SIMD8* writeback message with **Return Format** of 16-bit consists of up to 4 destination registers). Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the send instruction.

: If Pixel Null Mask Enable is enabled an additional register is included after the last channel register. Behavior of pixels that had a null texel contribution depends on the setting of **Null Pixel Behavior**.

DWord	Bits	Description
W0.7:4		Reserved
W0.3	31:16	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Half Float, S15 signed 2's comp integer, or U16 unsigned integer.
	15:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.2	31:16	**Subspan 1, Pixel 1 (upper right) Red**
	15:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.1	31:16	**Subspan 0, Pixel 3 (lower right) Red**
	15:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.0	31:16	**Subspan 0, Pixel 1 (upper right) Red**
	15:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W2		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W3		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written): This W4 is only delivered when **Pixel Fault Mask Enable** is enabled.
W4.0	31:24	Reserved: always written as 0xffff
W4.0	23:16	Reserved: always written as 0xff
W4.0	15:8	Reserved: always written as 0xff
W4.0	7:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

SIMD4x2

A SIMD4x2 writeback message always consists of a single message register containing all four channels of each of the two "pixels" (called "samples" here, as they are not really pixels) of data. The write channel mask bits as well as the execution mask on the "send" instruction are used to determine which of the channels in the destination register are overwritten. If any of the four execution mask bits for a sample is asserted, that sample is considered to be active. The active channels in the write channel mask will be written in the destination register for that sample. If the sample is inactive (all four execution mask bits deasserted), none of the channels for that sample will be written in the destination register.

: If Pixel Null Mask Enable is enabled an additional register is included after the last channel register. Behavior of pixels that had a null texel contribution depends on the setting of **Null Pixel Behavior**.

DWord	Bit	Description
W0.7	31:0	**Sample 1 Alpha:** Specifies the value of the pixel's alpha channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer.
W0.6	31:0	**Sample 1 Blue**

DWord	Bit	Description
W0.5	31:0	**Sample 1 Green**
W0.4	31:0	**Sample 1 Red**
W0.3	31:0	**Sample 0 Alpha**
W0.2	31:0	**Sample 0 Blue**
W0.1	31:0	**Sample 0 Green**
W0.0	31:0	**Sample 0 Red**
W1.7:1		Reserved (not written) : W1 is only delivered when **Pixel Fault Mask Enable** is enabled.
W1.0	31:2	Reserved: always written as 0x3fffffff
	1:0	**Pixel Null Mask:** This field has the bit for all samples set to 1 except those pixels in which a null page was source for at least one texel.

Direct Write To Render Target

The 3D Sampler supports the ability to return sample results directly to the Render Target rather than returning them to the requesting agent (e.g. the pixel shader). This feature allows the shader thread which issued the sample message to terminate after sending if the shader is complete. To use this capability, the shader must set the **Response Length** field in the message descriptor to 0 and it must send a message header which sets the **Render Target or Destination Binding Table Index** to the Binding Table Index of the Render Target.

There are a number of important restrictions and considerations:

- As described in the Message Header definition, sample message which require direct write to Render Target require a header. This increases the number of data phases in all message types and may impact performance in some cases.
- Direct write to Render Target by the sampler does not guarantee ordering of writes to the render target from different threads. Multiple threads writing to the same pixel using direct Render Target writes may cause serialization of these threads which may impact performance.

Programming Note	
Context:	Direct Render Target Writes
Sample operations which require direct writes to Render Target must not receive a page-fault response from memory.	

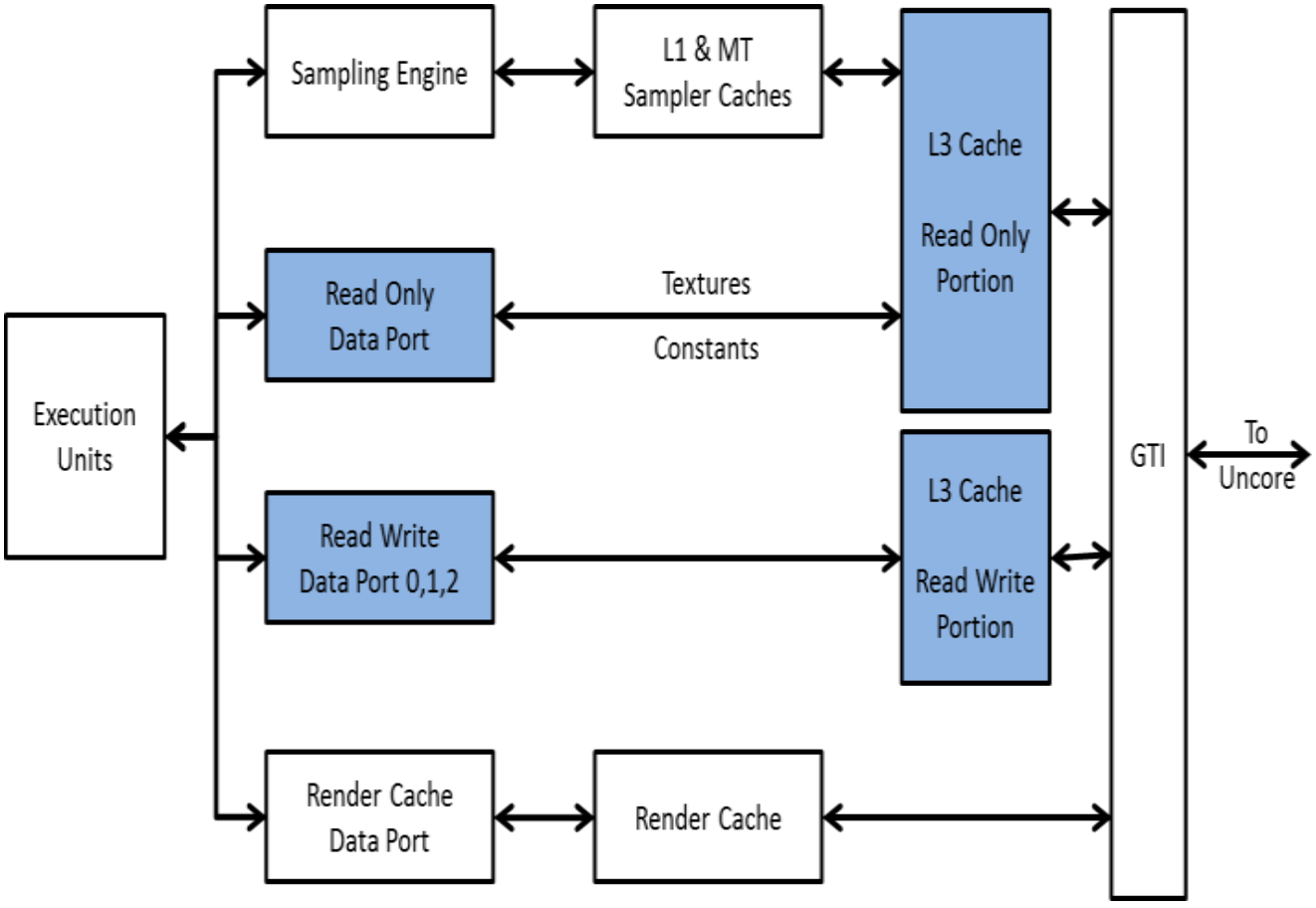
Programming Note	
Context:	SVM
Direct Write to Render Target does not support SVM memory model. Specifically, the Fault and Retry behavior is not supported for direct-write-to-rendert-target shaders.	

Shared Functions – Data Port

The Data Port provides all memory accesses for the Gen subsystem other than those provided by the sampling engine. These include constant buffer reads, scratch space reads/writes, and media surface accesses. Render Target accesses to the Render Cache are described in the *Shared Functions Render Target* chapter.

The diagram below shows how the Read-Only and Read/Write Data Ports connect with the caches and memory subsystem. The execution units and sampling engine are shown for clarity.

Data Port Connections to Caches and Memory



The kernel programs running in the execution units communicate with the data port via messages, the same as for the other shared function units. The data ports are considered to be separate shared functions, each with its own shared function identifier.

Read/Write Ordering

Memory access ordering in Gen subsystem programs is managed inside of each thread. Memory access ordering between threads is managed by software because threads can be run in any order, and another thread can run when a thread is blocked waiting for data.

Read, write, and atomic operations issued from the same thread to the same address, using the same data port, the same cache coherency type, and the same page faulting mode, are guaranteed to be processed in the same order as they are issued. Software mechanisms are used to ensure ordering of accesses when issued from different threads, to different addresses, with different data ports, with different cache coherency types, or with different page faulting modes.

So within the same thread, memory accesses may not be ordered under some circumstances . For example:

- The order that a single SIMD data port read or write message performs its multiple memory accesses is not guaranteed.
- If two different virtual addresses map to the same physical memory address, the order of the operations is not guaranteed.
- If a read misses a data cache and needs to fetch data from system memory, an access to a different address can complete earlier if it hits a data cache.
- If a virtual address faults, a different memory page access can proceed while the faulted access is handled.

Compared to the Fault-and-Halt page fault mode, the Fault-and-Stream page fault mode is a performance optimization that sometimes enables different order of accesses to continue after a page fault, and still guarantee the order of accesses to the same address within the same thread. Under some circumstances, the hardware automatically treats a Fault-and-Stream page fault as a Fault-and-Halt page fault. See Page Faults in Memory Views for more details on the behaviors of these modes. The Memory Fence message is used to block a thread until all pending Fault-and-Stream page faults are handled when ordering is required.

When memory access ordering is required and not guaranteed, either in the same thread or with different threads, software must ensure the ordering using fence operations or atomic operations. The Memory Fence message can be used to block a thread until all previous messages issued to that data port cache agent have been globally observed from the point of view of other threads in the system. Different threads in the same group can use a gateway barrier message with the "Flush to Global Observability" control to guarantee memory ordering at the point when all the threads have reached the barrier. Software can use the results of atomic operations to implement in programs a memory ordering between threads using exclusion locks or other protocols.

Programming Note	
Context:	Read/Write Ordering
In some configurations, a thread group that shares SLM memory can span multiple HDCs. When this occurs, SLM memory ordering is guaranteed by software using a either a memory fence operation or a message barrier	

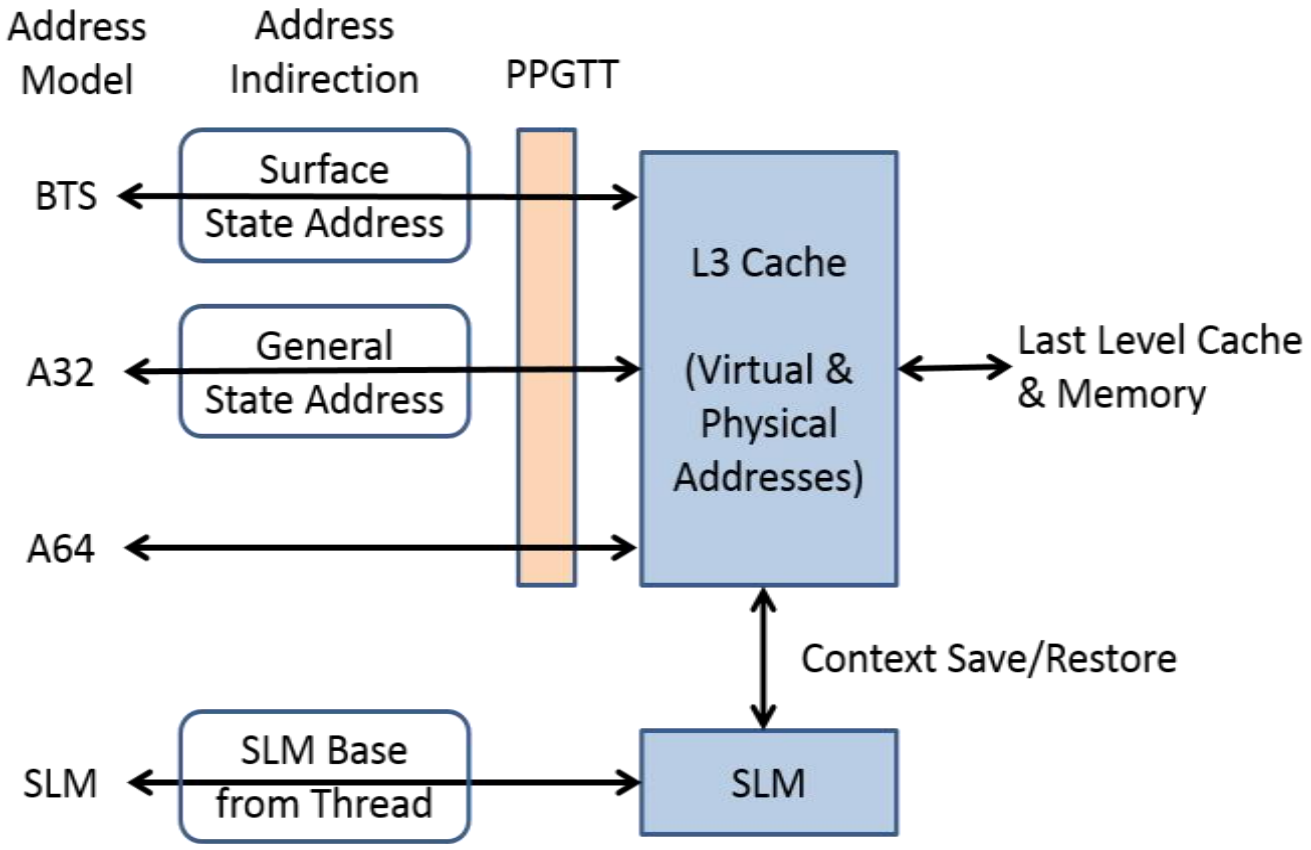
Programming Note	
Context:	Read/Write Ordering
operation to ensure global observability. In configurations where a thread group cannot span multiple HDCs (and when not running in Fault-and-Stream page fault mode), then SLM memory ordering is guaranteed without a fence.	

Address Models

Data structures accessed by the data port are called “surfaces”. There are four different Address Models used by the data port to access these surfaces: Binding Table State model (BTS), Shared Local Memory model (SLM), 32-bit Stateless model (A32), and 64-bit Stateless model (A64).

Most messages support only a subset of the Address Models.

Address Models



Surface State Addresses and General State Addresses are translated to virtual Graphics Addresses using the corresponding Base Address in the STATE_BASE_ADDRESS pointers, along with additional base address offset information provided with the Data Port message. SLM addresses are offset by the SLM Base Address that is set with each thread dispatch.

The sections below explain how the Base address is calculated for each Address Model. The Data Port messages then combine that Base address with multiple SIMD offsets for the message's memory operations.

Each calculated memory address is bounds-checked against the Address Model's boundaries. Out-of-bounds read accesses return zero and out-of-bounds write accesses are dropped.

Binding Table Surface State Model (SSM and BTS)

The data port uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine.

Surface State Address Model Description
The surface state model is used when a Binding Table Index (specified in the message descriptor) of less than 240 is specified. In this model, the Binding Table Index is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE.

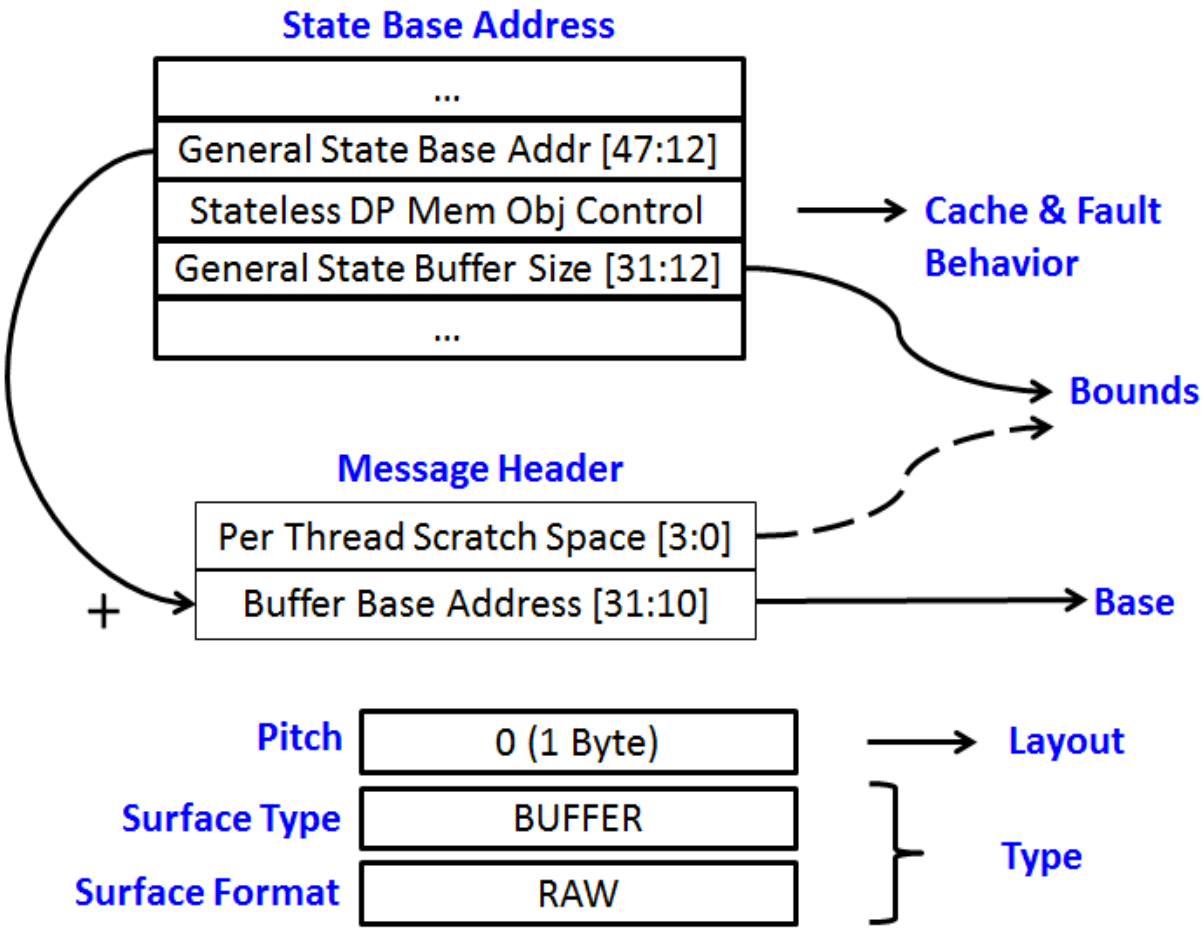
A32 Stateless Model

In the A32 model, the binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER
- Surface Format = RAW
- Vertical Line Stride = 0
- Surface Base Address = General State Base Address + Buffer Base Address
- Buffer Size = compared against General State Buffer Size for bounds checking
- Surface Pitch = 1 byte
- Utilize Fence = false
- Tiled = false

This model is primarily intended for scratch space buffers and for GPGPU global memory accesses.

A32 Stateless Memory Characteristics



Stateless Model Aliases

The stateless aliases 255 and 253 provide a means of SW controlling the coherency properties of an access. The coherency property is ensured for that access only. Typically, SW uses the same coherency type for all access to the same address. Proper fencing is required to ensure that reads and writes are visible. L3UC forces the addressed cache lines out of L3 and the cycles are directly conducted to LLC. This provides a capability for ensuring coherency on a particular location without having to fence all the other cycles.

Binding Table Index	Type	Description
255	IA Coherent	Coherent within Gen and coherent within the entire IA cache memory hierarchy.
253	Non-Coherent	Coherent within Gen for the same Gen cache type (L3, Sampler, or Render).

Programming Note	
Context:	A32 Stateless Model
• The constant, sampler, and render caches are always non-coherent. • A32 messages that are stateless and specified without a BTI (Hword Read/Write and all messages on SFID_DP_DC2) are IA coherent.	

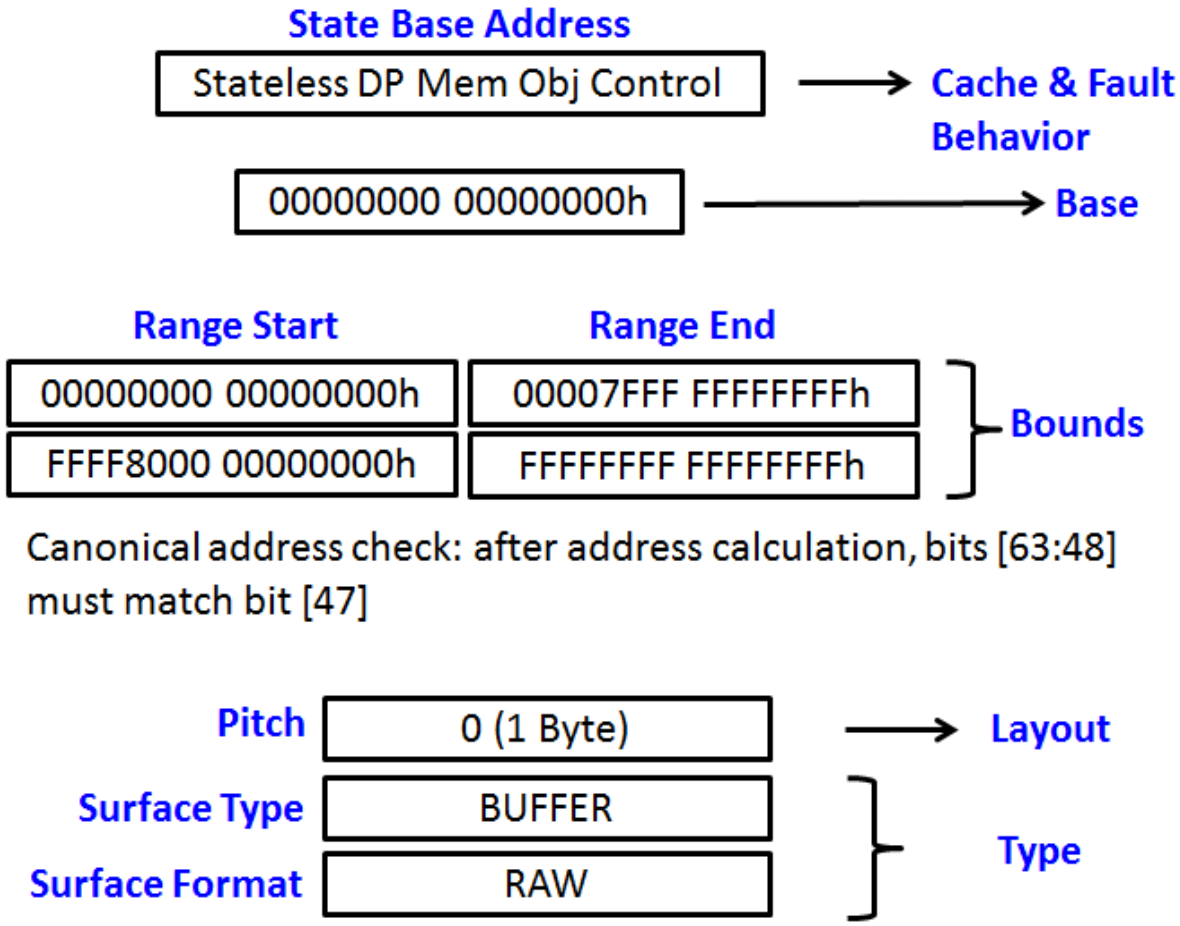
A64 Stateless Model

In the A64 model, the binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER
- Surface Format = RAW
- Vertical Line Stride = 0
- Surface Base Address = Graphics Address
- Buffer Size = not checked
- Surface Pitch = 1 bytes
- Utilize Fence = false
- Tiled = false

This model is primarily intended for programmable shader programs.

A64 Stateless Memory Characteristics



Programming Note	
Context:	A64 Stateless Model
A64 messages that are stateless and specified without a BTI (Hword Read/Write and all messages on SFID_DP_DC2) are IA coherent.	

Shared Local Memory (SLM)

Shared local memory (SLM) is a high bandwidth memory that is not backed up by system memory. It is enabled by configuring the L3 cache to use part of its space for the SLM. The SLM contents are shared between all active threads in the same thread group. Its contents are uninitialized after creation, and its contents disappear when deallocated.

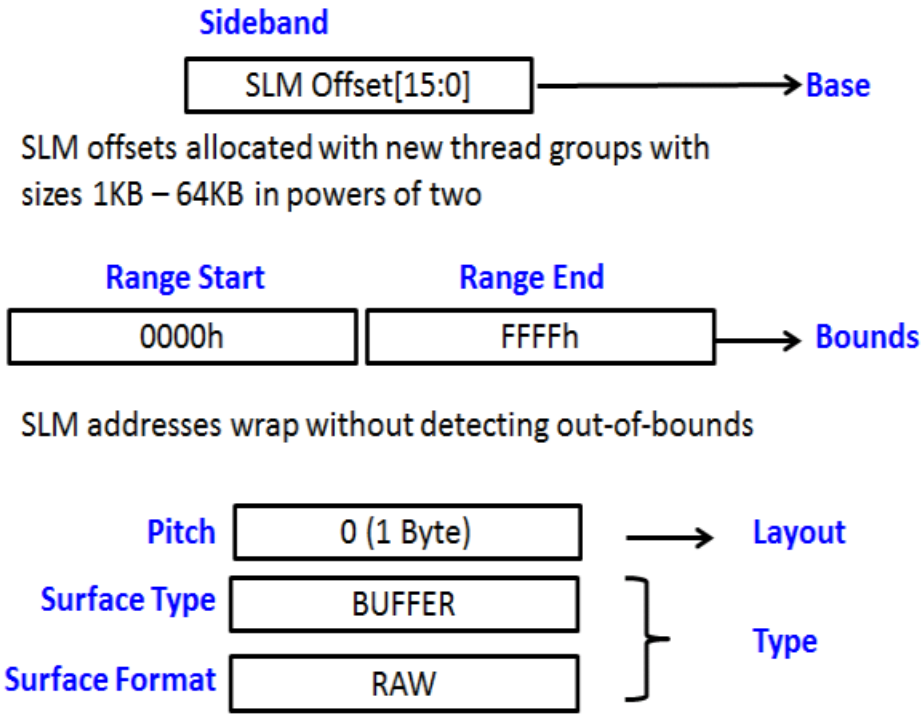
In the SLM model, the binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER

- Surface Format = RAW
- Surface Base Address = points to the start of the internal SLM (no memory address is applicable)
- Surface Pitch = 1 byte

Due to the predefined surface state attributes for the SLM, only a subset of the data port messages can be used.

SLM Memory Characteristics



SLM is accessed when a Binding Table Index (specified in the message descriptor) of 254 is specified, or by Data Port 2 messages that are not accessing A32 or A64 memory.

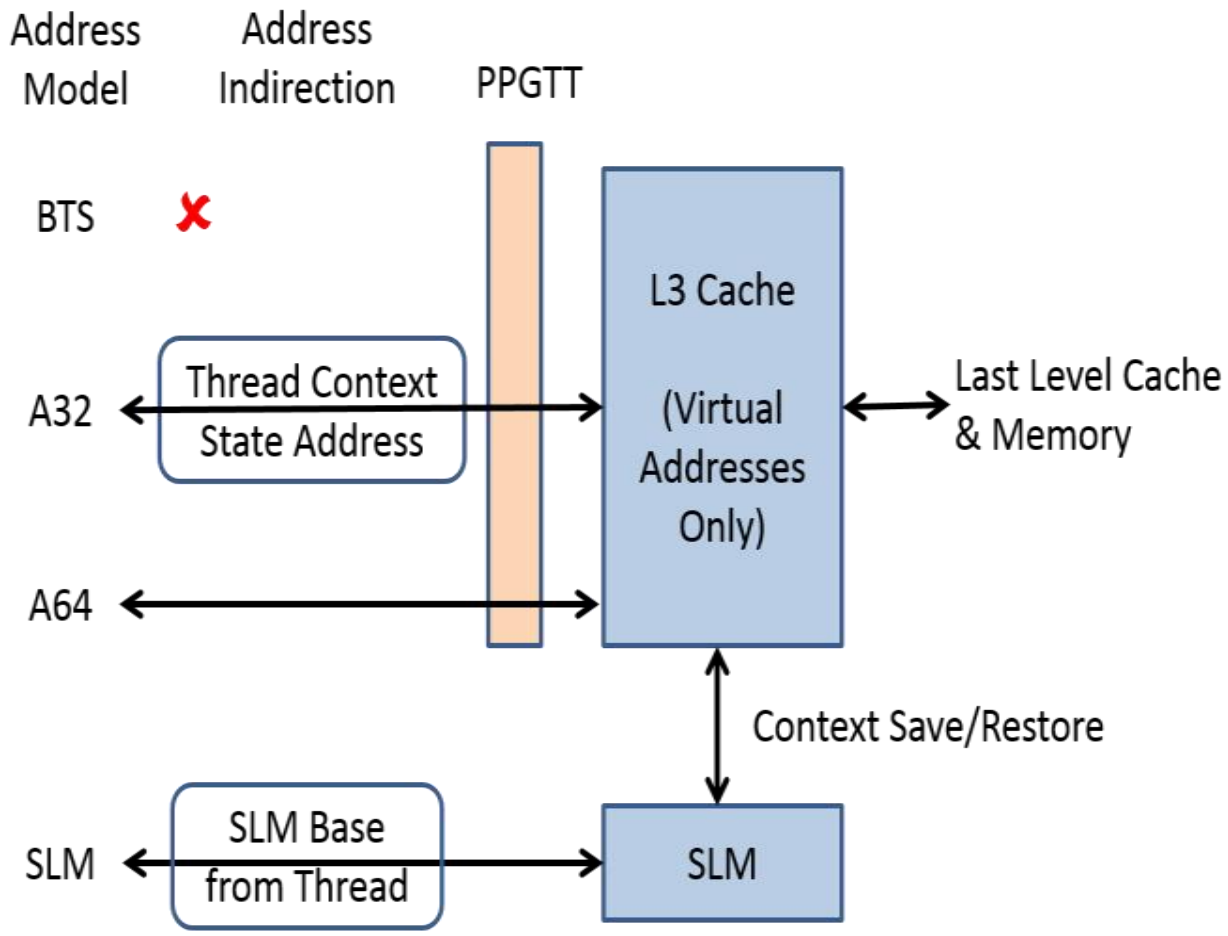
Programming Note	
Context:	Shared Local Memory (SLM)
• Only the data cache data port is supported for SLM, the other data ports treat Binding Table Index 254 as a normal surface state access. • Accesses to SLM do not have any bounds checking. Addresses beyond the size (64 KB) of the SLM wrap around.	

SLM should not be accessed through the 3D pipe.

Address Models during SIP

When the SIP routine for the thread is running, only the stateless and SLM address models are supported with Data Port messages.

Address Models during SIP



The A32 base address is overridden during the SIP execution to be the thread's context save area.

Stateless A32 and A64 memory accesses are overridden to be non-coherent (L3 cache stores them as virtual addresses).

In addition to the address model restrictions during the SIP execution, only a subset of Data Port messages are defined for use during the SIP execution.

Context Save and Restore Supported Messages for SIP
Hword Scratch Read/Write messages are the only messages that override A32 base address with the Context Save area's base address during SIP.

Overview of Memory Accesses

The specific message operation modifies the characteristics of the Address Model to determine the behaviors of each Data Port memory access. An overview of the characteristics is provided in the table below, and then explained in more detail in the following sections.

Characteristic	Overview
Surface Type	Each message has its own requirements for which surface types are supported. Most messages only operate on a restricted subset of surface types. Most Data Port messages only operate on the SURFTYPE_BUFFER surface type, but some specific messages operate on other structured surface types.
Address Alignment	Each message has its own requirements for address alignment. Most messages require an address that is aligned on a natural boundary for the width of the data item being accessed.
Address Calculation	Each message calculates one common Base Offset, and multiple additional Offsets for the SIMD operation. Depending on the message, the SIMD Offsets can be byte address offsets, or indices that select an address aligned by the message operation's data width.
Bounds Checking	The per-message Base Address Offset is added to each of the message's additional SIMD Offsets to form offsets. Those offsets are each checked that they are within the surface's and message's boundaries before a memory access is made. If a memory access is within bounds, each calculated offset is added to the Address model's Base Address to form a virtual, physical, or SLM address that is accessed through the L3 cache structure. If a memory access is out-of-bounds, then the message-specific action occurs (see Bounds Checking in this volume).
Canonical Address Checking	The calculated virtual memory address, formed from each calculated offset and the Address model's Base Address, could cross the canonical address boundary. Messages using Binding Table State and A32 stateless address models require the graphics driver to correctly set up the base and bounds parameters so that every in-bounds access does not cross the canonical address boundary. Messages using A64 Stateless address models detect any calculated offsets that cross the canonical address boundary, either as a fault or as an out-of-bounds access.
Tiled Resources Checking	The virtual memory address for a data port access may be in the range of addresses that are recognized as Tiled Resources, and that are translated through the TRTT (Tiled Resources Translation Table). Sparse Tiled Resources support a kind of application-managed virtual memory scheme, where the application tells the driver to map specified 64KB address regions within the surface to memory resources called Tile Pools. Tiles that are not mapped to a Tile Pool are either: • Null tiles, which are treated as Out-of-Bounds

Characteristic	Overview
	• Invalid tiles, which are treated as page faults Tiles that are mapped to a Tile Pool have a virtual address that is handled like any virtual address that is outside of the Tiled Resources address range.
Fault Handling	Virtual memory addresses for data ports are translated through the PPGTT. If a fault occurs in either the fault-and-stream or fault-and-halt page fault modes, a fault handler is invoked on the IA processor to potentially correct the fault condition and retry the access. Execution of a faulted thread is suspended until the IA driver signals the GPU to restart all faulted threads and replay the faulted memory accesses. SLM accesses do not fault. See Read/Write Ordering in this volume for a description of how memory ordering is impacted by a page fault.
Out-of-Bounds handling	If a memory access is marked as out of bounds for the surface or the message, or if a virtual address maps to a Tiled Resource Null page, then a read operation returns 0 and a write operation is dropped.
Cache Behavior	Data port memory values can be cached both inside or outside the GPU. The primary GPU data cache can either be kept coherent with the IA processor with every memory acess, or left non-coherent where software protocols manage the coherency. Each Address Model specifies the coherency model for the read/write data cache ports. • For Surface State accesses: the surface's RENDER_SURFACE_STATE **Coherency Type** field • For A32 and A64 Stateless accesses: the 253 and 255 aliases (see Stateless Model Aliases in A32 Stateless Model in this volume for details) • SLM accesses are localized and not cached The behavior of writing cache data to memory (e.g. write back or write through modes) is specified by the MEMORY_OBJECT_CONTROL_STATE. • For Surface State accesses: the surface's RENDER_SURFACE_STATE **Surface Object Control State** • For A32 and A64 Stateless accesses: the STATE_BASE_ADDRESS **Stateless Data Port Access Memory Object Control** When PPGTT is enabled, the page table entry overrides the LLC/eDRAM cache controls supplied by the MEMORY_OBJECT_CONTROL_STATE.

Surfaces Types

Each message has its own requirements for which surface types are supported. Most messages only operate on a limited subset of surface types. The BUFFER surface type is supported by all the address models. All other surface types are only supported by the BTS address model.

The stateless and SLM address models have a pre-defined surface type, format, and layout. Untyped messages are used with these address models.

The BTS address model is used with either typed or untyped messages. The BTS surface type must be compatible with the expected surface type, format, and layout of the message operation. Untyped messages override the surface format with their specific data type and size. Typed messages require the surface characteristics expected by the message, or the message results are undefined (accesses may be dropped or may return random data).

Surface Types used by Data Port Messages

Message Category	BTS	SLM	A32	A64
Render Target	1D, 2D, 3D, CUBE, NULL	N/A	N/A	N/A
Media	2D, NULL	N/A	N/A	N/A
Typed Surface or Atomic	1D, 2D, 3D, CUBE, BUFFER, NULL	N/A	N/A	N/A
Untyped Surface or Atomic	BUFFER, STRBUF, NULL	BUFFER	BUFFER	BUFFER
Untyped Scattered or Block	BUFFER, NULL	BUFFER	BUFFER	BUFFER
Scaled Untyped Scattered or Surface	N/A	BUFFER	BUFFER	BUFFER

When a surface state is present, untyped messages ignore the surface format and treat the data as RAW. For example, data is returned from the constant buffer to the GRF without format conversion.

2D, 3D, CUBE and NULL surfaces support YMAJOR tile modes to improve memory locality of nearby accesses. All surface types support LINEAR tile mode. Surfaces with type BUFFER and STRBUF cannot be tiled. 2D surfaces support XMAJOR tile mode for the Display engine. Stencil format is supported in WMAJOR tile mode for 2D, 3D and CUBE tile formats.

Tile Modes supported by Data Port Surface Types

Surface Type	Linear	Y Major	W Major	X Major
BUFFER	Yes			
STRBUF	Yes			
1D	Yes			
2D	Yes	Yes	Yes	Yes
CUBE	Yes	Yes	Yes	
3D	Yes	Yes	Yes	
NULL	Yes	Yes	Yes	Yes

See Surface Layout and Tiling in the Memory Views volume for more information about tile layouts.

Programming Note	
Context:	Surfaces Types
For data port messages, compressed surface formats for MSAA and Media surfaces must be Y Major tiled (not Linear, X Major, or W major).	

Addressing Buffers (SURFTYPE_BUFFER)

Most data port messages operate on BUFFER surface types. The figure below illustrates how the base address and bounds calculations are performed on BUFFER surface types.

BUFFER: Array of Structures

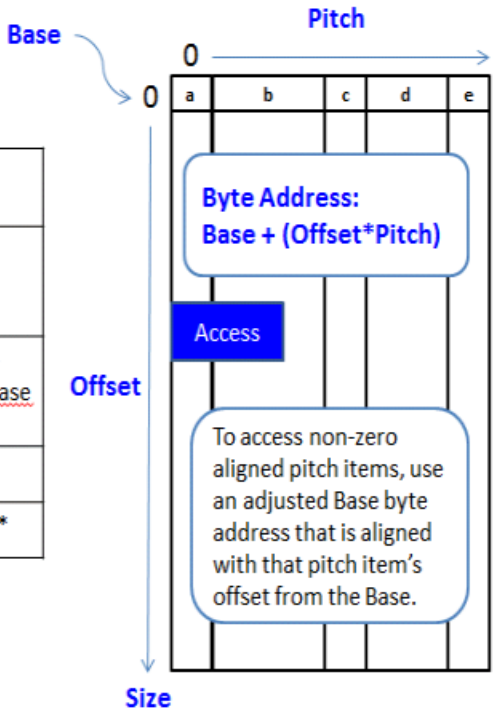
Bounds Check:
- **Offset < (Buffer Size – ((Access Width - 1) / Pitch))**

Address Model	BTS	A32	A64	SLM
Base Address Alignment	32B	2KB	0	1KB
Buffer Size	Surface Size*	General State Buffer Size	Canonical = 47 bits	64KB – SLM_Base
Offset Size	32 bits	32 bits	64 bits	32 bits
Pitch Size	11 bits	16 bits**	16 bits**	7 bits**

* Surface Size - 1 =

Depth	Height	Width
U11	U14	U7

** Pitch Size specified in Data Port 2 scaled messages



For untyped messages accessing SURFTYPE_BUFFER surface types, the U address (byte offset) must be aligned to the data access width (for example, DWords must have the low 2 bits as zeros).

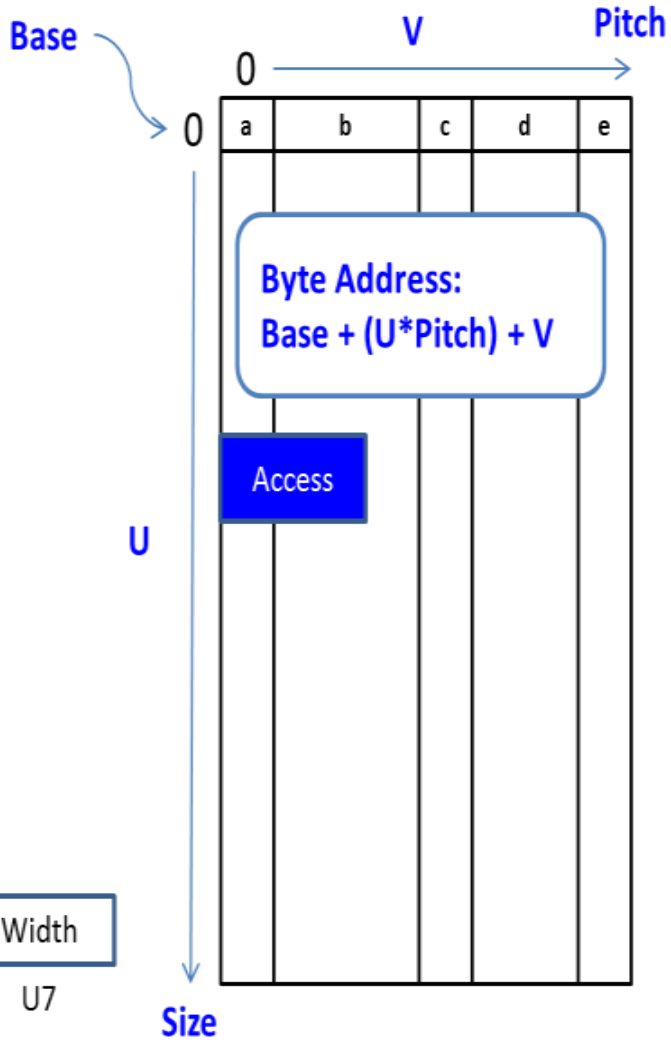
Addressing Structured Buffers (SURFTYPE_STRBUF)

Some data port messages operate on STRBUF surface types. The figure below illustrates how the base address and bounds calculations are performed on BUFFER surface types.

STRBUF: Array of Structures

Bounds Check:
- **U < Size**
- **V < Pitch – Access Width**

Address Model	BTS
Base Address Alignment	32B
Buffer Size	Surface Size*
U & V Size	32 bits
Pitch Size	11 bits



* Surface Size - 1 =

Depth	Height	Width
U11	U14	U7

Programming Note	
Context:	Addressing Structured Buffers (SURFTYPE_STRBUF)
Read/Write Data Ports only support linear tiled STRBUF.	

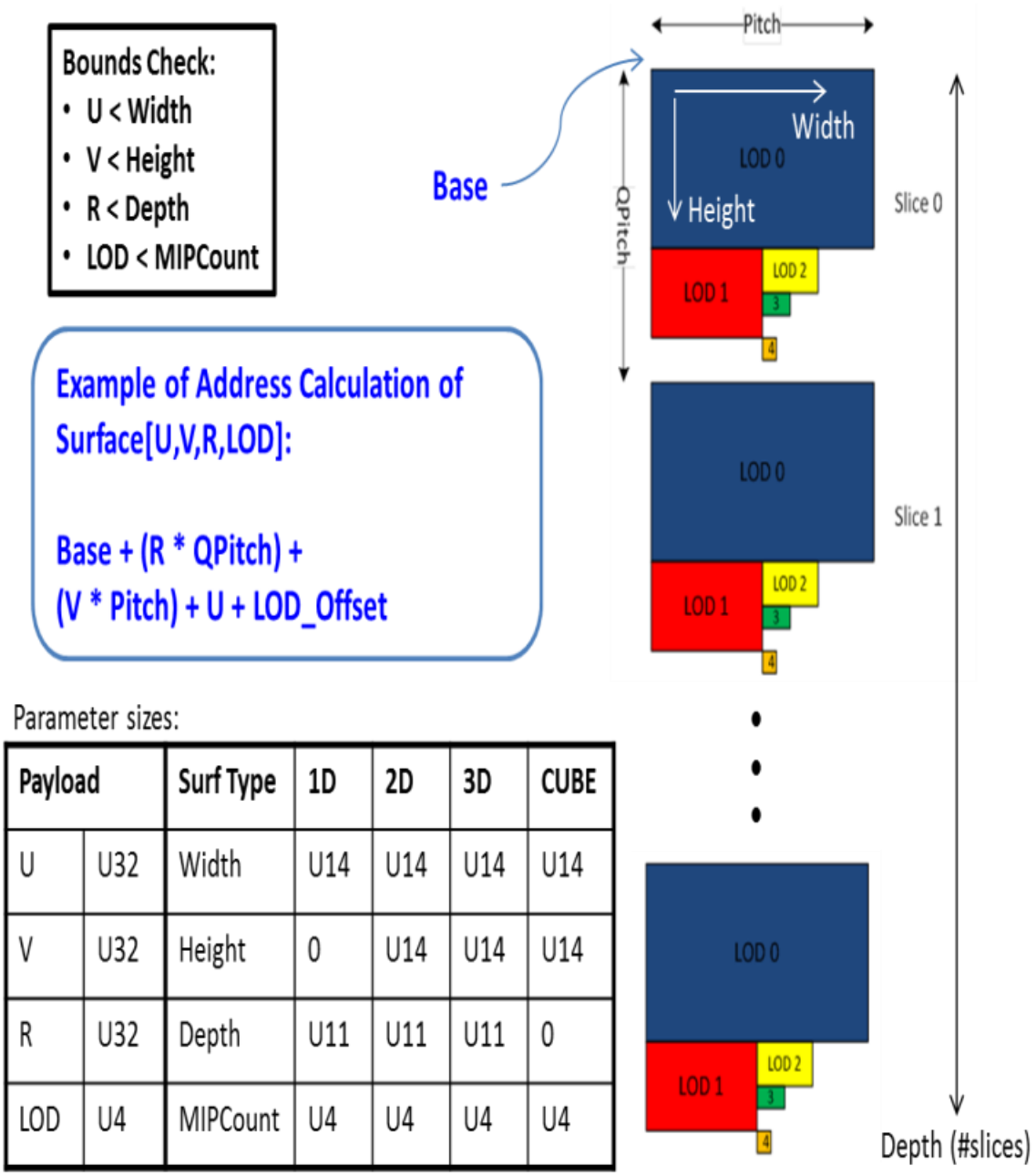
For untyped messages accessing SURFTYPE_STRBUF surface types, both the V address and Surface Pitch must be aligned to the data access width (for example, DWords must have the low 2 bits as zeros).

Addressing 1D, 2D, 3D, CUBE Surfaces

Some data port messages operate on various BTS surface types. The addressing and bounds checking for the 1D, 2D, 3D, and CUBE surface types is fully explained in other chapters. See SURFACE_STATE in the Shared Functions volume and Surface Layout and Tiling in the GPU Overview volume.

The figure below illustrates the differences in addressing and bound checking between these surface types and the BUFFER and STRBUF surface types.

Multi-Dimensional Surface Buffers



Parameter sizes:

Payload		Surf Type	1D	2D	3D	CUBE
U	U32	Width	U14	U14	U14	U14
V	U32	Height	0	U14	U14	U14
R	U32	Depth	U11	U11	U11	0
LOD	U4	MIPCount	U4	U4	U4	U4

The following table indicates the hardware interpretation of each input address parameter based on surface type. If LOD is specified, then both V and R parameters must also be included, even if they are ignored. If LOD is not specified, then unused V and R parameters are optional in the address payload. If LOD is not present in the address payload, then it has the default value of 0. If the surface state indicates the Number of Multisamples > 1, then the sample number is the first parameter in the address payload.

Address Parameters Used by Surface Type

SURFACE_STATE Fields			Address Payload				
Surface Type	Surface Array	Number of Multisamples	MSAA	U Address	V Address	R Address	LOD
SURFTYPE_1D	disabled	Must be MULTISAMPLECOUNT_1	Never Present	X pixel address	(ignored if present)	(ignored if present)	Optional LOD
	enabled			X pixel address	array index	(ignored if present)	Optional LOD
SURFTYPE_2D	disabled	MULTISAMPLECOUNT_1	Never Present	X pixel address	Y pixel address	(ignored if present)	Optional LOD
	enabled			X pixel address	Y pixel address	array index	Optional LOD
	disabled	Not MULTISAMPLECOUNT_1	Sample Number	X pixel address	Y pixel address	(ignored if present)	Optional LOD
	enabled			X pixel address	Y pixel address	array index	Optional LOD
SURFTYPE_CUBE	disabled	Must be MULTISAMPLECOUNT_1	Never Present	X pixel address	Y pixel address	array index	Optional LOD
	enabled			X pixel address	Y pixel address	array index	Optional LOD
SURFTYPE_3D	disabled	Must be MULTISAMPLECOUNT_1	Never Present	X pixel address	Y pixel address	Z pixel address	Optional LOD
SURFTYPE_BUFFER	disabled	Must be MULTISAMPLECOUNT_1	Never Present	buffer index			
SURFTYPE_STRBUF	disabled			buffer index	byte offset		

Workaround	
Context:	Addressing 1D, 2D, 3D, CUBE Surfaces

If the surface state indicates the Number of Multisamples > 1, then the LOD parameter is not optional: the R and LOD parameters must be specified along with the MSAA sample number parameter.

Programming Note	
Context:	Addressing 1D, 2D, 3D, CUBE Surfaces

- Vertical stride & Vertical Offset fields of the surface state object are only supported for 2D non-array surfaces.
- Tile W surfaces must be of format R8_UINT and only support SIMD8.

Surface Formats

Untyped messages allow direct read and write access to untyped surfaces. Untyped surfaces are always type BUFFER or STRBUF, and are assumed to be surface format RAW.

Typed messages allow direct read and write accesses to typed surfaces. Typed surfaces are of type SURFTYPE_1D, 2D, 3D, CUBE, or BUFFER and have a supported surface format other than RAW.

Read and write messages for typed surfaces convert the message's data type to/from the selected surface's format. The conversion rules are listed in a table below.Some surface formats are supported for typed surface writes, but are indirectly supported for typed surface reads. Indirectly supported surface formats provide the RAW pixel data in UINT formats, which enables software to complete the conversion to their defined format.

The read conversion of typed surface format to a register data format is always exact. If the destination precision is less than the source precision, a write conversion from a register data format to a typed surface format may be rounded, and denormalized or clamped.

Conversion Rules for Typed Surface Dataport Messages

Register Data Type	Surface Format Type	Read Conversion (Surface to Register)	Write Conversion (Register to Surface)
F32	FLOAT	IEEE float conversion. Normalize denorms if source is narrower. Convert SNan to QNan quietly.	IEEE float conversion. Round to even and denormalize if destination is narrower. Convert SNan to QNan quietly.
F32	SNORM, UNORM	Convert fixed point to IEEE float	Convert IEEE float to fixed point. Round to even and clamp to min/max if destination is narrower.
S31	SINT	Sign extend MSBs if source is narrower	Clamp to min/max if destination is narrower.
U32	UINT	Zero extend MSBs if source is narrower	Clamp to min/max if destination is narrower.
RAW32	Any	32 bit copy	*Not supported*
RAW16	Any	16 bit copy, zero extend MSBs	*Not supported*
RAW8	Any	8 bit copy, zero extend MSBs	*Not supported*

Typed dataport messages support a subset of all surface state formats. The formats supported by typed surface reads and writes are listed in the table below. The table shows what values and formats are returned for the components when read.

Typed surface formats with UINT require the message data in U32 format. Surface formats with SINT require the message data in S32 format. All other typed surface formats require the message data in FLOAT32 format. The surface format for the typed atomic integer operations must be R32_UINT or R32_SINT.

All typed surface formats are supported for writes, as long as the Bits Per Element is a multiple of 8 (byte). This includes all the surface formats excluding SRGB and YCRV formats. The table shows the full list of supported surface formats for writes.

For typed surface writes where the Surface Format has components that are not byte-aligned, each shader channel select in the surface state must be set to a unique surface channel (SCS_RED, SCS_GREEN, SCS_BLUE, SCS_ALPHA) and the value of (SCS_ZERO, SCS_ONE) cannot be selected. Also all channels must be enabled for writing.

Reads to an unsupported surface format return undefined results. Writes to an unsupported surface format have undefined results.

When an out-of-bounds location is read by typed dataport message, a 0 is returned for any component that is present in the surface format, and the default value is returned for any component that is missing in the surface format. The default value for missing components is 0 for RGB, and either 1 or 1.0 for A.

Supported Typed Surface Write and Read Formats

Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)	Data Returned by Typed Read			
			R	G	B	A
000	R32G32B32A32_FLOAT	128	F32	F32	F32	F32
001	R32G32B32A32_SINT	128	S31	S31	S31	S31
002	R32G32B32A32_UINT	128	U32	U32	U32	U32
080	R16G16B16A16_UNORM	64	RAW32	RAW32	0	1
081	R16G16B16A16_SNORM	64	RAW32	RAW32	0	1
082	R16G16B16A16_SINT	64	S31	S31	S31	S31
083	R16G16B16A16_UINT	64	U32	U32	U32	U32
084	R16G16B16A16_FLOAT	64	F32	F32	F32	F32
085	R32G32_FLOAT	64	F32	F32	0.0	1.0
086	R32G32_SINT	64	S31	S31	0	1
087	R32G32_UINT	64	U32	U32	0	1
0C0	B8G8R8A8_UNORM	32	*RAW32*	0	0	1
0C2	R10G10B10A2_UNORM	32	*RAW32*	0	0	1
0C4	R10G10B10A2_UINT	32	*RAW32*	0	0	1
0C7	R8G8B8A8_UNORM	32	RAW32	0	0	1
0C9	R8G8B8A8_SNORM	32	RAW32	0	0	1
0CA	R8G8B8A8_SINT	32	S31	S31	S31	S31
0CB	R8G8B8A8_UINT	32	U32	U32	U32	U32
0CC	R16G16_UNORM	32	RAW32	0	0	1
0CD	R16G16_SNORM	32	RAW32	0	0	1
0CE	R16G16_SINT	32	S31	S31	0	1
0CF	R16G16_UINT	32	U32	U32	0	1
0D0	R16G16_FLOAT	32	F32	F32	0.0	1.0
0D1	B10G10R10A2_UNORM	32	*RAW32*	0	0	1
0D3	R11G11B10_FLOAT	32	*RAW32*	0	0	1

Surface Format Encoding (Hex)	Format Name	Bits Per Element (BPE)	Data Returned by Typed Read			
			R	G	B	A
0D6	R32_SINT	32	S31	0	0	1
0D7	R32_UINT	32	U32	0	0	1
0D8	R32_FLOAT	32	F32	0.0	0.0	1.0
100	B5G6R5_UNORM	16	*RAW16*	0	0	1
102	B5G5R5A1_UNORM	16	*RAW16*	0	0	1
104	B4G4R4A4_UNORM	16	*RAW16*	0	0	1
106	R8G8_UNORM	16	RAW16	0	0	1
107	R8G8_SNORM	16	RAW16	0	0	1
108	R8G8_SINT	16	S31	S31	0	1
109	R8G8_UINT	16	U32	U32	0	1
10A	R16_UNORM	16	RAW16	0	0	1
10B	R16_SNORM	16	RAW16	0	0	1
10C	R16_SINT	16	S31	0	0	1
10D	R16_UINT	16	U32	0	0	1
10E	R16_FLOAT	16	F32	0.0	0.0	1.0
11A	B5G5R5X1_UNORM	16	*RAW16*	0	0	1
140	R8_UNORM	8	RAW8	0	0	1
141	R8_SNORM	8	RAW8	0	0	1
142	R8_SINT	8	S31	0	0	1
143	R8_UINT	8	U32	0	0	1
144	A8_UNORM	8	RAW8	0	0	1

Programming Note	
Context:	Surface Formats
Indirectly supported surface formats are treated as R8_UINT for the 8 BPE formats, R16_UINT for 16 BPE formats, R32_UINT for 32 BPE formats, and R32G32_UINT for 64 BPE formats. And the Channel Mask for surface read and write messages, and the Shader Channel Select fields from the surface state, are interpreted with these same substitute formats.	

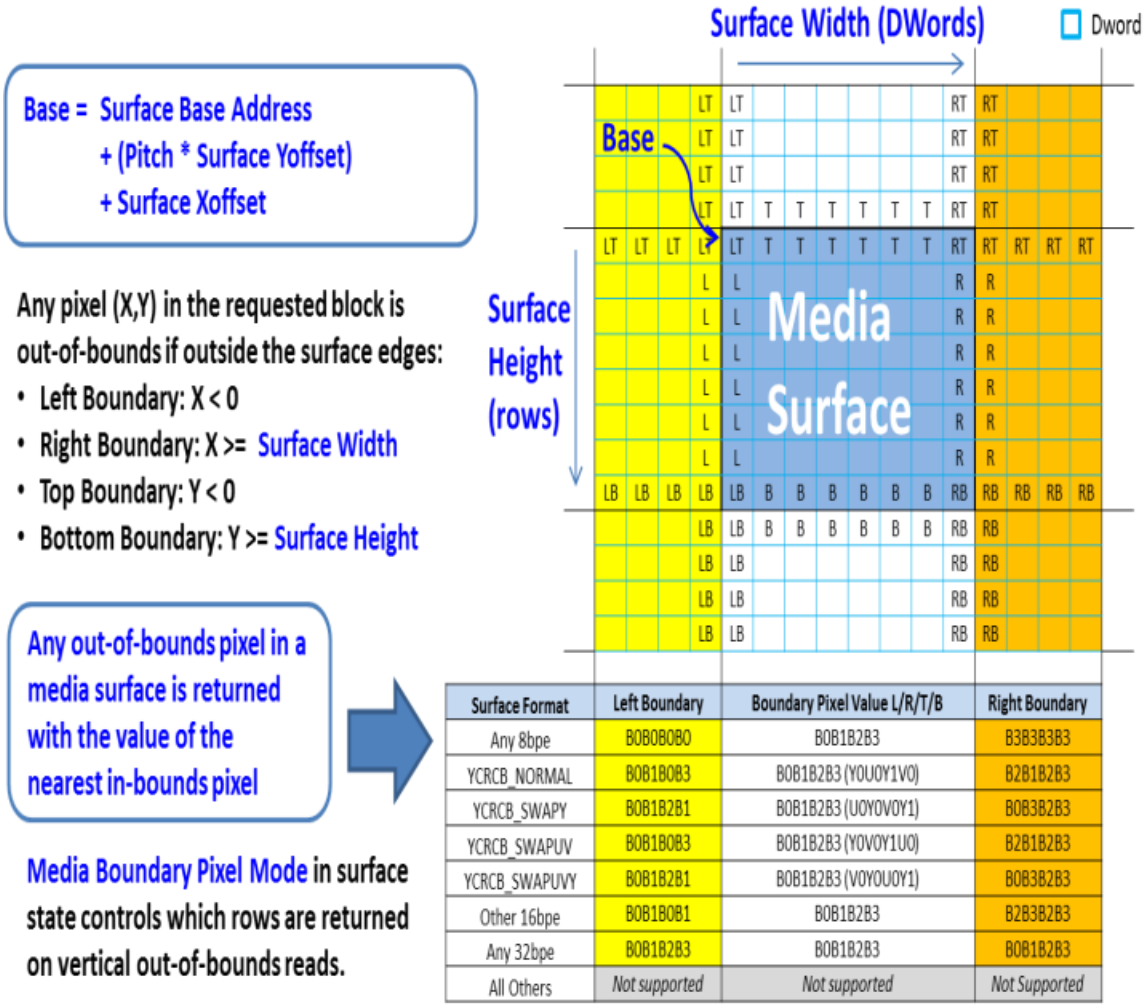
Addressing 2D Media Surfaces

Some data port messages operate on rectangular blocks of 2D surfaces using the BTS address model. These messages are typically used in media image processing. The rectangular block is specified in media messages as a signed (X, Y) offset from the surface's origin, and by the block's width and height.

The rectangular block specified in a media message could be partially (or fully) out-of-bounds of the surface. Some media data port messages perform pixel replication to supply values for out-of-bounds

pixels. The figure below illustrates how the boundary pixel values are calculated from an in-bounds pixel value.

Media Surface Boundary Pixel Handling



Surface Format	Left Boundary	Boundary Pixel Value L/R/T/B	Right Boundary
Any 8bpe	B0B0B0B0	B0B1B2B3	B3B3B3B3
YCRCB_NORMAL	B0B1B0B3	B0B1B2B3 (Y0U0Y1V0)	B2B1B2B3
YCRCB_SWAPY	B0B1B2B1	B0B1B2B3 (U0Y0V0Y1)	B0B3B2B3
YCRCB_SWAPUV	B0B1B0B3	B0B1B2B3 (Y0V0Y1U0)	B2B1B2B3
YCRCB_SWAPUVY	B0B1B2B1	B0B1B2B3 (V0Y0U0Y1)	B0B3B2B3
Other 16bpe	B0B1B0B1	B0B1B2B3	B2B3B2B3
Any 32bpe	B0B1B2B3	B0B1B2B3	B0B1B2B3
All Others	*Not supported*	*Not supported*	*Not Supported*

Media accesses are Dwords: all 4 surface boundaries are Dword aligned
(Base is 32B aligned, Pitch is 64B aligned, Xoffset is 4*bpe aligned, Width is Dword aligned).

Upper left corner of the requested Block is specified by the signed (X, Y) coordinate.
The block offset and block width are always aligned to size of pixel (BPE) in surface.

The addressing of 2D Media surfaces is subset of the generalized surface state addressing model. For more information, see SURFACE_STATE in the Shared Functions volume and Surface Layout and Tiling in the GPU Overview volume.

The Vertical Stride and Vertical Stride Offset fields of the surface state object are used for 2D media surfaces to support interlaced formats in a full frame.

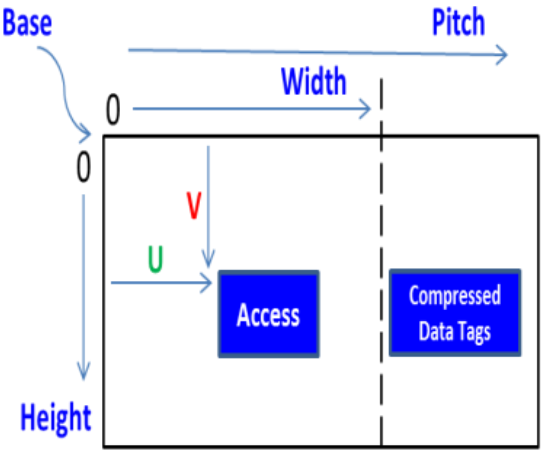
The surface's tile mode specifies how the 2D surface is laid out in the linear virtual address space. The figure below illustrates how pixel's byte address is calculated in a 2D media surface. The table lists how the surface's linear virtual address offset A[38:0] is calculated.

Media Surface Addressing

Guarantee the access is in surface bounds
with Media Boundary Pixel Handling:

- Clamp X in [0, Width-1]
- Clamp Y in [0, Height-1]
- U = Surface Xoffset + (4 * Clamp X)
- V = Surface Yoffset + Vertical Line Stride Offset
  + (Vertical Line Stride * Clamp Y)

Byte Address: Surface_Base + Tiled_2D_Surface_Address_Offset



Media surfaces are optionally compressed, with compressed data tags in pitch region

Bits per Element	Tile Mode	Tile constants		Tiled 2D Surface Address Offset																	
		Cv	Cu	A[38:16]	A[15:12] (64KB Tile)				A[11:6] (4KB Tile)						A[5:0] (Cache Line)					
64 & 128	TileYS	6	10	u[13:10] + (v[13:6]*Pitch[17:10])	u9	v5	u8	v4	u7	v3	u6	v2	u5	u4	v1	v0	u3	u2	u1	u0
	TileYF	4	8	u[13:8] + (v[13:4]*Pitch[17:8])																
16 & 32	TileYS	7	9	u[13:9] + (v[13:7]*Pitch[17:9])	u8	v6	u7	v5	u6	v4	u5	v3	u4	v2						
	TileYF	5	7	u[13:7] + (v[13:5]*Pitch[17:7])																
8	TileYS	8	8	u[13:8] + (v[13:8]*Pitch[17:8])	u7	v7	u6	v6	u5	v5	u4	v4	u3	v2						
	TileYF	6	6	u[13:6] + (v[13:6]*Pitch[17:6])																
all	TileY	5	7	u[13:7] + (v[13:5]*Pitch[17:7])					u6	u5	u4	v4	u3	v2						
all	TileX	3	9	u[13:9] + (v[13:3]*Pitch[17:9])					v2	v1	v0	u8	u7	u6	u5	u4	u3	u2	u1	u0
all	Linear	0	0	u[13:0] + (v[13:0]*Pitch[17:0])																

When a 2D surface is tiled, the Surface Base Address is aligned on the tile size (4KB or 64KB).
NV12 surfaces use X/Y Offset for UV Plane surface state so that Luma 8bpe and Chroma 16bpe both aligned for TileYF/YS.

2D Media surfaces can be set up to be compressible using the Memory Compression Enable surface state field. Some media hardware will write blocks in a compressed format. The media dataport messages will read those compressed formats and automatically decompress a compressed location when read. The tag information needed to interpret and decompress data is stored in the memory region between the surface width and pitch.

Programming Note	
Context:	Addressing 2D Media Surfaces
If the surface state field Memory Compression Enable is set, then the surface state field Coherency Type must be set to GPU Coherent (not IA coherent).	

Programming Note	
Context:	Addressing 2D Media Surfaces
Writing a compressible media surface with any dataport message is not supported and produces undefined results. Reading a compressible media surface using any dataport message other than Media Block Read is not supported and produces undefined results.	

Programming Note	
Context:	Addressing 2D Media Surfaces

- The surface base address must be 32-byte aligned.
- The surface width must a multiple of DWords.
- Pitch must be a multiple of 64 bytes when the surface is Linear.
- Media block writes to Linear or TileX surfaces must have a height of 16 or less.
- For YUV422 formats, the block width and offset must be pixel pair aligned (i.e. DWord-aligned).
- For media block writes, both X Offset and Block Width must be DWord-aligned.
- The block width and offset must be aligned to the size of pixels stored in the surface. For a surface with 8bpe pixels for example, the block width and offset can be byte-aligned. For a surface with 16bpe pixels, it is word-aligned.

2D Media Surface Formats

Media dataport messages allow direct read and write accesses to media surfaces. Media surfaces are only type SURFTYPE_2D.

Read and write messages for media surfaces never convert the data type of the selected surface's format.

If the media surface state is set up as not compressible, the formats supported are listed in the table below. Reads or writes to an unsupported surface format have undefined results. The surface format is only used to determine out-of-bounds pixel replication.

If the media surface state is set up as compressible, then only a small subset of surface state formats are supported by the media data port message. The formats supported by compressible media surface reads are listed in the table below. Reads to an unsupported surface format return undefined results. Writes to a compressible surface format have undefined results. The surface format is used to determine out-of-bounds pixel replication and to determine compression type.

Supported Media Surface Formats

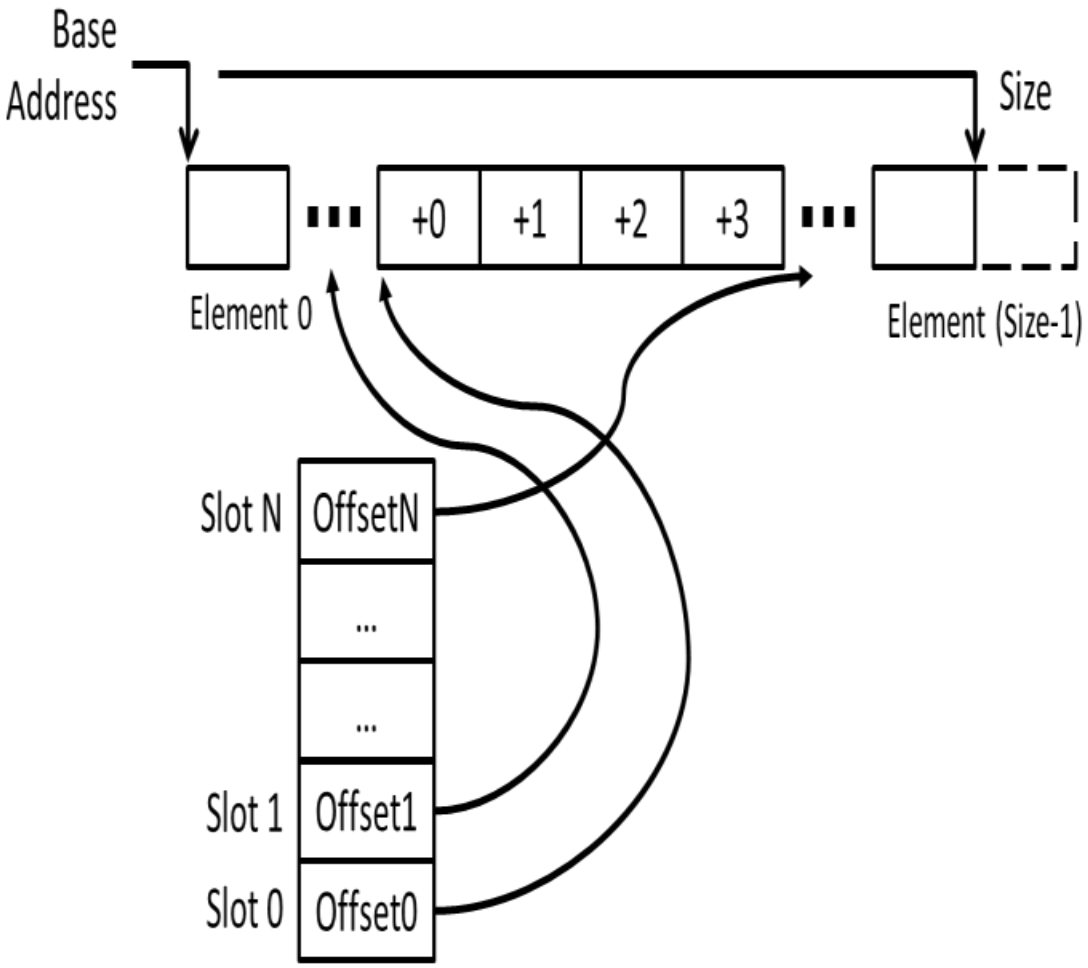
Surface Format Encoding (Hex)	Format Name	Bits Per Element(BPE)	Supported Compressible Media Surface Formats
0C0	B8G8R8A8_UNORM	32	*Not supported*
0C2	R10G10B10A2_UNORM	32	*Not supported*
0C4	R10G10B10A2_UINT	32	*Not supported*
0C7	R8G8B8A8_UNORM	32	*Not supported*
0C9	R8G8B8A8_SNORM	32	*Not supported*
0CA	R8G8B8A8_SINT	32	*Not supported*
0CB	R8G8B8A8_UINT	32	*Not supported*
0CC	R16G16_UNORM	32	*Not supported*
0CD	R16G16_SNORM	32	*Not supported*
0CE	R16G16_SINT	32	*Not supported*
0CF	R16G16_UINT	32	*Not supported*
0D0	R16G16_FLOAT	32	*Not supported*
0D1	B10G10R10A2_UNORM	32	*Not supported*
0D3	R11G11B10_FLOAT	32	*Not supported*
0D6	R32_SINT	32	*Not supported*
0D7	R32_UINT	32	*Not supported*
0D8	R32_FLOAT	32	*Not supported*
100	B5G6R5_UNORM	16	*Not supported*
102	B5G5R5A1_UNORM	16	*Not supported*
104	B4G4R4A4_UNORM	16	*Not supported*
106	R8G8_UNORM	16	NV12 (UV 8bpe)
107	R8G8_SNORM	16	*Not supported*
108	R8G8_SINT	16	*Not supported*
109	R8G8_UINT	16	NV12 (UV 8bpe)
10A	R16_UNORM	16	Y16
10B	R16_SNORM	16	*Not supported*
10C	R16_SINT	16	*Not supported*
10D	R16_UINT	16	Y16
10E	R16_FLOAT	16	*Not supported*
113	A16_UNORM	16	Y16
11A	B5G5R5X1_UNORM	16	*Not supported*
140	R8_UNORM	8	NV12 (Y), Y8, IMC1 (YUV), IMC3 (YUV)
141	R8_SNORM	8	*Not supported*
142	R8_SINT	8	*Not supported*

Surface Format Encoding (Hex)	Format Name	Bits Per Element(BPE)	Supported Compressible Media Surface Formats
143	R8_UINT	8	NV12 (Y), Y8, IMC1 (YUV), IMC3 (YUV)
144	A8_UNORM	8	NV12 (Y), Y8, IMC1 (YUV), IMC3 (YUV)
182	YCRCB_NORMAL	16	YUYV
183	YCRCB_SWAPUVY	16	VYUY
18F	YCRCB_SWAPUV	16	YVYU
190	YCRCB_SWAPY	16	UYVY

Slots and Elements

Data Port messages are SIMD operations: a single instruction operating over multiple data operands. Data operands are addressed either as: consecutive offsets (“elements”), with arbitrary offsets (“slots”), or in a combination of both slots and elements.

Slots and Elements



Each data port message supports a restricted subset of all possible combinations of slots and elements. The details of which combinations are supported are listed for each message in the Messages section of this volume.

The Offset in slots can be in any order and may be duplicates. The data port hardware attempts to minimize memory accesses and optimize for cache line efficiency. The hardware issues in parallel the data operations in parallel for each slot and data element pair, up to the limits of the hardware implementation. Bounds checking is performed independently on each access.

Programming Note	
Context:	Slots and Elements
Writes to overlapping addresses have undefined write ordering.	

For A32 messages, a base address offset is provided in the message header and added to each offset. The base address offset in the message header is always ignored for BTS, SLM, and A64 messages.

Typed messages always use the BTS address model, and can use multiple address components to calculate the address offset for each slot. The Surface Type determines which address components are present and how the address offset is calculated.

Base Address Offsets in SIMD

Slots	SIMD Name	Location of Slot Address Offset	Description
1	Block	Message Header	Message header provides 1 address offset in that is used as the address offset of a block of data elements. Untyped blocks use 1 address. Typed blocks such as Media Block use X/Y coordinates.
2	Dual Block or SIMD4x2	Address Payload	Message address payload provides 2 address offsets which are used as the base address offset of 2 sets of data elements.
8	SIMD8	Address Payload	Message address payload provides 8 address offsets which are used as the address offset of 8 sets of data.
16	SIMD16	Address Payload	Message address payload provides 16 address offsets which are used as the address offset of 16 sets of data. Typed messages process either the upper or lower 8 slots at a time, so two messages must be used for SIMD16 operations.

Programming Note	
Context:	Slots and Elements
The current implementation limits the maximum data payload for a message to 256 bytes. Depending on the number of slots and the data width specified by the operation, some messages restrict the number of elements supported in a single SIMD operation.	

Execution Masks

Every data port message is sent with a 16-bit execution mask. The Execution Mask is used by the data port to enable the read or write operation on the mask's corresponding SIMD channel. The channel is considered active if the mask bit is set.

The execution mask is used to ensure that data associated with an inactive channel are not overwritten in the GRF (read operations) or memory (write operations).

Data port messages have different semantics on how the execution mask controls the read or write accesses. The table below summarizes the semantics for all the data ports. The details of which semantics are used by each individual message is described in the Messages section of this volume.

Description of SIMD Execution Mask Semantics

Execution Mask Name	SIMD Mode and Data Type	How Execution Mask is Used
SM	SIMD8 or SIMD16	For SIMD16, the 16 bits of the execution mask control the 16 SIMD channels. For SIMD8, the lower 8 bits of the execution mask control the 8 SIMD channels, and the upper 8 bits of the execution mask are ignored.
SMBLK	SIMD8 or SIMD16 With Data Elements	For SIMD16, the 16 bits of the execution mask control the 16 SIMD channels. For SIMD8, the lower 8 bits of the execution mask control the 8 SIMD channels, and the upper 8 bits of the execution mask are ignored. If the date elements > 1, then the execution mask bits are reused for the additional data elements.
SMPSM	SIMD8 or SIMD16 Pixel Surface	For SIMD16, the 16 bits of the execution mask control the 16 SIMD channels. For SIMD8, the lower 8 bits of the execution mask control the 8 SIMD channels, and the upper 8 bits of the execution mask are ignored. The bits of the execution mask are ANDed with the corresponding bits of the Pixel/Sample Mask from the message header, and the resulting mask determines which slots are enabled. If the message header is not present, only the execution mask is used.

Execution Mask Name	SIMD Mode and Data Type	How Execution Mask is Used
SM4x2	SIMD4x2	The lower 8 bits of the execution mask control the 2 SIMD channels. The lower 4 bits control the first channel and the upper 4 bits control the upper channel. The upper 8 bits are ignored. For reads and writes, if any one or more bits within the corresponding execution mask nibble are enabled, the channel is enabled.
SG	SIMD8 Slot Group	Either the lower 8 bits or the upper 8 bits of the execution mask are used to control the 8 SIMD channels, based on the slot group specified in the message descriptor.
SGPSM	SIMD8 Slot Group on Pixel Surface	Either the lower 8 bits or the upper 8 bits of the execution mask are used to control the 8 SIMD channels, based on the slot group specified in the message descriptor. The bits of the execution mask are ANDed with the corresponding bits of the Pixel/Sample Mask from the message header, and the resulting mask determines which slots are enabled. If the message header is not present, only the execution mask is used.
BLK	Block	The lower 8 bits of the execution mask control the first 8 DWords, and the upper 8 bits control the second 8 DWords. For reads, the enables operate in OWord channel mode: if any one or more asserted bits within the OWord's corresponding execution mask nibble are enabled, the whole OWord is read regardless of the setting of any particular DWord's bit. For writes, the enables operate in DWord channel model: the DWords are written if the corresponding DWord's execution mask bit is set. If the block is 1 OWord size, then either the lower or upper nibbles of the lower 8 bits control the corresponding DWords in that lower or upper OWord. If the block is 8 OWords or 16 OWords, then the execution mask bits are reused for the additional channels.
DBLK	Dual Block	The lower 8 bits of the execution mask control the 8 SIMD DWord channels. The lower 4 bits control the first channel and the upper 4 bits control the second channel. The upper 8 bits are ignored. For reads, the enables operate in OWord channel mode: if any one or more asserted bits within the OWord's corresponding execution mask nibble are enabled, the whole OWord is read regardless of the setting of any particular DWord's bit. For writes, the enables operate in DWord channel model: the DWords are written if the corresponding DWord's execution mask bit is set. If the block is 8 OWords (4 dual blocks), then the lower 8 execution mask bits are

Execution Mask Name	SIMD Mode and Data Type	How Execution Mask is Used
		reused for the additional OWord blocks 1-3.
BLKCM	Block with Channel Mode	The lower 8 bits of the execution mask control the first 8 DWords, and the upper 8 bits control the second 8 DWords. For DWord channel mode, the execution mask delivered with the message dictates DWord-based control of read or write operations. For OWord channel mode, any one or more asserted bits within the OWord's corresponding execution mask nibble causes read or write operations to occur across all four DWords of the OWord regardless of the setting of any particular DWord's bit. If the block is 1 OWord size, then either the lower or upper nibbles of the lower 8 bits control the corresponding DWords in that lower or upper OWord. If the block is 8 OWords or 16 OWords, then the execution mask bits are reused for the additional blocks.
Ignored	Not Used	The execution mask bits are ignored by this message operation. The data is controlled by other parameters to this message. The execution mask bits must be non-zero for the message to be sent to the data port.

Address Alignment and Data Widths

Data Port messages calculate multiple data operand addresses using offsets from the message's common base address. The details of how each message calculates its address is described in the Messages section.

The address calculations are described using the following notational convention:

(Base + BaseOffset) → DataType[Offset]

This notation refers to the byte-aligned address (**Base + BaseOffset**), which is used as the address of a linear array of **DataType** operands. The **Offset** is the index into the **DataType** array with that data width.

Address Calculation

Notation	Description
Base	Each Address Model has its own base address source, illustrated in the Surfaces Types section of this volume. The Base address is architecturally defined to always be aligned at a valid address boundary for the message. The notational convention in this chapter labels this calculated base address the **Base**.

Notation	Description
Buffer, BaseOffset	The offsets from some messages are specified as byte offsets, and others are DWord (or other DataType) offsets. **Buffer** is shorthand for Buffer Base Address offset that is specified in A32 message headers. The notation in this chapter shows byte offsets as being added to the **Base** on the left side of the pointer (→), and the DataType-sized offsets as being indices to the array on the right of the pointer. Each message defines the address alignment it operates on. Base address offsets are forced to be aligned for the message's DataType by treating any unaligned low address bits as zero. For most messages (except media block messages), the Buffer or BaseOffset value is always an unsigned (positive) value.
B[ ], W[ ], DW[ ], QW[ ], OW[ ], HW[ ]	The **DataType** array is a linear array of Bytes (**B**), Words (**W**), DWords (**DW**), QWords (**QW**), OWords (**OW**), or HWords (**HW**). The index of the array is multiplied by the width of the data type, to form a byte offset to be added to the base address. Each message defines the data type it operates on. The array's byte offset is architecturally defined to always be aligned at a valid address boundary for the message.
Offset{Slot}, U{Slot}, V{Slot}	**Offset{Slot}** is shorthand for Offset0, Offset1, … in a SIMD8 or SIMD16 address payload. **U{Slot}** and **V{Slot}** are shorthand for saying U.Offset{Slot} and V.Offset{Slot}, where U and V are SIMD8 or SIMD16 address payloads. For most messages (except media block messages), the **Offset** is always an unsigned (positive) value.
Elem, Chan	**Elem** represents the index into an array of consecutive data operands. **Chan** represents the 32-bit Red, Green, Blue, or Alpha channels of color item. These values are sequentially stored in memory as a 4 item DW block.
(Surface[U, V, R, LOD])	The notation **Surface[U, V, R, LOD]** refers to a typed surface which, depending on the surface type, may use the U, V, R, and LOD parameters to calculate the offsets from the surface's state base address for the operands.

Bounds Checking and Faulting

Bounds checking is a programming safety feature of the Gen architecture. If a data port access is outside the boundaries of the surface, that memory operation is dropped before accessing the L3 cache. Out-of-bounds read operations return zero for the data. Each SIMD memory access is individually checked in a message operation, so some accesses could be in-bounds and while others are out-of-bounds.

Out-of-bounds checking is always performed at a DWord granularity. If any part of the DWord is out-of-bounds then the whole DWord is considered out-of-bounds.

The bounds check is performed on the Address Offset portion of the message operation. It is designed to catch user mode programming mistakes. The bounds check assumes that the surface's base address is properly aligned and that the surface's boundary limits are properly configured by the trusted kernel mode graphics driver.

The bounds check does not replace the data security features of the page tables (PPGTT). The page tables are used to protect other memory areas from accesses outside of this program's memory space. Virtual memory addresses for data ports are translated through the PPGTT, and follow that translation table's faulting model. In all page fault modes, either the fault condition is corrected and the memory access is completed, or the entire program is terminated.

Tiled Resources
If a data port access is made to a TRTT Null tile, then that access behaves like an out-of-bounds access.

The bounds checks performed are specific to the address model and surface type of the message. See the figures in the Surfaces Types section of this volume for more information about the boundary conditions for the surfaces.

Description of Boundary Check Semantics

Bounds Check	Address Model	Surface Type	Description
Surface	BTS	BUFFER	Messages using the BTS address model detect any address offsets that, either directly or through the offset calculations, exceed the offset size of the buffer as specified in the SURFACE_STATE.
Surface	BTS	STRBUF	Messages using the BTS address model detect any U address component that exceeds the U size of the structured buffer, or a V address component that exceeds the Pitch of the structured buffer, as specified in the SURFACE_STATE.
Surface	BTS	1D, 2D, 3D, CUBE	Messages using the BTS address model detect any U, V, or R address components that exceed the Width, Height, and Depth sizes of the surface, as specified in the SURFACE_STATE.
Media	BTS	2D	Some media-oriented messages using the BTS address model detect X or Y address components that exceed the Width and Height of the surface, as specified in the SURFACE_STATE. If a read access is outside of the surface boundaries, the read access is treated as an in-bounds access and returns the value of the nearest pixel. Write accesses outside of the surface boundaries are dropped.
GenState	A32	BUFFER	Messages using the A32 Stateless address model detect any address offsets that, either directly or through the offset calculations, exceed the General State Buffer Size specified in the STATE_BASE_ADDRESS. If General State Buffer Size is zero, then any A32 Stateless access is out-of-bounds.
PTSS	A32	BUFFER	Some A32 messages specify a Per Thread Scratch Space Size in their message header. When that message header is present and Local Thread Checking is enabled in **GT Mode Register**, the A32 message detects any address offsets that,

Bounds Check	Address Model	Surface Type	Description
			directly or through the offset calculations, exceed the Per Thread Scratch Space Size. These A32 messages must additionally perform the GenState boundary check because the PTSS value is not guaranteed by trusted software to be within the GenState boundary limits.
Canonical	A64	BUFFER	Messages using A64 Stateless address model detect any calculated address offsets that cross the canonical address boundary as out-of-bounds. Any A64 offsets that are used directly as the byte address without scaling do not require a bounds check. Messages using non-A64 address models do not need to check for crossing the canonical address boundary because their surface and boundaries are assumed to be properly configured by the graphics driver.
Wrap	SLM	BUFFER	Messages using the SLM address model do not detect any address offsets that, either directly or through the offset calculations, exceed the SLM memory size. If the SLM memory size is exceeded, the SLM address wraps around in the SLM memory and is treated as an in-bounds access.

Message Formats

Message operations on data ports are described using five stardard parts. The message's Descriptor is sent to the target data port on a message sideband bus, and the other four parts are sequentially transmitted between the data port and EU registers across the OBUS. The total number of registers sent and received by the data port is provided to the data port on the message sideband bus.

Descriptor	Describes the Message Type, the Message Specific Controls, and whether a Message Header is present. The Message Descriptor and the destination Data Port (SFID) is encoded as part of the SEND instruction.
Header	When present, provides additional Message specific controls. Some messages disallow (forbid) a Header, some require a Header, and some permit an optional Header to specify additional non-default-valued parameters.
Address Payload	Provides the slot address offsets for those messages that support scattered operations.
Source Payload	Provides source data for write operations and atomic operations.
Writeback Payload	Returns result data for read operations and for atomic operations.

Most messages do not use (and therefore do not send or receive) all five parts of a message. The specific message encoded in the Message Descriptor determines which of the other four parts of the message sequence are present.

The next sections describe all of the supported formats for the Source and Writeback Data Payloads, the Address Payloads, the Message Descriptors, and their Message Headers.

End of Thread Usage

Read/Write data port messages may not have the End of Thread bit set in the message descriptor other than the following exceptions:

The Media Block Write message may have End of Thread set for pixel shader threads dispatched by the windower in contiguous dispatch mode.

Message Description Conventions

Message operations with common semantics and limitations are grouped together and described in the same section. These common characteristics are summarized in the first table in each section:

Common Characteristics of a Message Grouping

Although multiple messages are described in the same section, they frequently perform their similar functions using different data ports, address models, and with different SIMD combinations of slots and blocks. The second table in each message section lists all the valid combinations of message parameters and payloads for the messages:

Description of Message Summary Tables

Column Heading	Description
Addr Align	Specifies the required address alignment of the operation. This can be byte (B), DWord (DW), QWord (QW), OWord (OW), or HWord (HW). In most cases, the address must be aligned for the data width of the operation.
Data Width	Specifies the width of the data operation. This can be byte (B), DWord (DW), QWord (QW), OWord (OW), or HWord (OW).
R/W	Specifies if the operation is a Read or Write (R or W). Atomic operations that return data are classified here as reads, and ones that don't are classified here as writes.
Address Model	Specifies the address model used by this message operation: BTS, SLM, A32, or A64. Read-only data port accesses to the constant cache or sampler cache are labeled "CC BTS" or "SC BTS".
Surface Type	Specifies the supported surface type. See the Surfaces Types section of this volume.
SIMD Slots	In the first table, this summarizes all the SIMD slot configurations supported for this operation and address model. In the second table, this specifies the specific SIMD slot configuration supported by the specific message operation.
Data Elements	Specifies all the sizes of data elements supported by the message operation. For blocks, this is the number of blocks. For surfaces, this is the list of channels that are enabled in the message descriptor. See MDC_CMASK Channel Mask Message Descriptor Control Field in this volume.

Column Heading	Description
SIMD Address Calculation	Describes the address calculation performed by this message operation. The calculation is usually related to the others in this section, but the details can change due to the address model used. See Address Alignment and Data Widths for an explanation of the notation used here.
Bounds Check	Specifies the bounds check performed by this message operation. The bounds check is generally determined by the address model used, but some message change the checking. See Bounds Checking and Faulting for an explanation of the notation used here.
Execution Mask	Most message operations perform their operation on a data channel only when the corresponding execution mask is set. See Execution Masks for an explanation of the notation used here.
Message Specific Descriptor	Each data port specifies a unique encoding for its message operations and its parameters. See Message Specific Descriptors for the full list of specific message descriptors used by the data ports. In each section, all the legal combinations of messages and parameters are listed. In some cases, there are multiple message operations that perform the same function.
Message Header	Many messages use a message heder to provide additional parameters to control their operation, in addition to the parameters in the message descriptor. Each section's table lists which message headers are legal to use. See Message Headers for a full list of the message headers that are supported by all the messages.
Address Payload	Most messages perform SIMD operations using address offsets for the operation from the address payload. See Message Address Payloads for a full list of the message address payloads that are supported by all the messages. Each section's table lists which message headers are legal to use.
Source Payload	Write messages and atomic messages generally provide data in a source data payload. See Message Data Payloads for a full list of the message data payloads that are supported by all the messages. Each section's table lists which message source payloads are legal to use.
Writeback Payload	Read messages generally return data in a writeback data payload. See Message Data Payloads for a full list of the message data payloads that are supported by all the messages. Each section's table lists which message writeback payloads are used.

Messages

The message operations on data ports are described this section. The operations are organized into subsections by their addressing operation group (blocks, scattered, surfaces, atomic operations), and then by their data width and type. Each subsection describes the messages that have common semantics and limitations. (For example, read and write operations are described in the same subsection.)

Summary of Message Groups

Operation	Addr Align	Data Width	Address Model	Surface Type	SIMD Slots	Data Elements	Bounds Check	Execution Mask
Scattered and Block R/W					1, 2	1, 2, 4, 8		
Byte Scattered R/W	B	B	BTS	BUFFER, NULL	8, 16	1, 2, 4	Surface	SMBLK
	B	B	SLM	BUFFER	8, 16	1, 2, 4	Wrap	SMBLK
	B	B	A32	BUFFER	8, 16	1, 2, 4	PTSS	SMBLK
	B	B	A32	BUFFER	8, 16	1, 2, 4	GenState	SMBLK
	B	B	A64	BUFFER	8, 16	1, 2, 4	Canonical	SMBLK
Byte Scaled R/W	B	B	SLM	BUFFER	8, 16	1, 2, 4	Wrap	SMBLK
	B	B	A32	BUFFER	8, 16	1, 2, 4	GenState	SMBLK
	B	B	A64	BUFFER	8, 16	1, 2, 4	Canonical	SMBLK
DW Scattered R/W	DW	DW	BTS	BUFFER, NULL	8, 16	1	Surface	SM
	DW	DW	A32	BUFFER	8, 16	1	PTSS	SM
	DW	DW	A64	BUFFER	8, 16	1, 2, 4, 8	Canonical	SMBLK
DW Scaled R/W	DW	DW	A64	BUFFER	8, 16	1, 2, 4, 8	Canonical	SMBLK
QW Scattered R/W	QW	QW	A64	BUFFER	8, 16	1, 2, 4	Canonical	SMBLK
QW Scaled R/W	QW	QW	A64	BUFFER	8, 16	1, 2, 4	Canonical	SMBLK
OW Block R/W	OW	OW	BTS	BUFFER, NULL	1	1, 2, 4, 8	Surface	BLK
	OW	OW	A32	BUFFER	1	1, 2, 4, 8	PTSS	BLK
	OW	OW	A64	BUFFER	1	1, 2, 4, 8	Canonical	BLK
OW Aligned Block R/W	DW	OW	BTS	BUFFER, NULL	1	1, 2, 4, 8	Surface	Ignored
	DW	OW	A32	BUFFER	1	1, 2, 4, 8	PTSS	Ignored
	DW	OW	A64	BUFFER	1	1, 2, 4, 8	Canonical	Ignored
OW Dual Block R/W	OW	OW	BTS	BUFFER, NULL	2	1, 4	Surface	DBLK
	OW	OW	A32	BUFFER	2	1, 4	PTSS	DBLK
	OW	OW	A64	BUFFER	2	1, 4	Canonical	DBLK
HW Block R/W	HW	HW	A32	BUFFER	1	1, 2, 4, 8	PTSS	BLKCM
HW	HW	A64	BUFFER	1	1, 2, 4, 8	Canonical	BLKCM	
Surface R/W					2, 8, 16	1		
Scattered Untyped Surface R/W	DW	OW	BTS	BUFFER, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	BUFFER, NULL	8, 16	{Chan 1-4}	Surface	SMPSM
	DW	OW	SLM	BUFFER	4x2	1	Wrap	SM4x2
	DW	DW	SLM	BUFFER	8, 16	{Chan 1-4}	Wrap	SMPSM
	DW	OW	A32	BUFFER	4x2	1	GenState	SM4x2

Operation	Addr Align	Data Width	Address Model	Surface Type	SIMD Slots	Data Elements	Bounds Check	Execution Mask
	DW	DW	A32	BUFFER	8, 16	{Chan 1-4}	GenState	SMPSM
	DW	OW	A64	BUFFER	4x2	1	Canonical	SM4x2
	DW	DW	A64	BUFFER	8, 16	{Chan 1-4}	Canonical	SM
Scaled Untyped Surface R/W	DW	OW	BTS	STRBUF, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	STRBUF, NULL	8, 16	{Chan 1-4}	Surface	SMPSM
	DW	OW	SLM	BUFFER	4x2	1	Wrap	SM4x2
	DW	DW	SLM	BUFFER	8, 16	{Chan 1-4}	Wrap	SM
	DW	OW	A32	BUFFER	4x2	1	GenState	SM4x2
	DW	DW	A32	BUFFER	8, 16	{Chan 1-4}	GenState	SMPSM
	DW	OW	A64	BUFFER	4x2	1	Canonical	SM4x2
	DW	DW	A64	BUFFER	8, 16	{Chan 1-4}	Canonical	SM
Scattered Typed Surface R/W	B	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	4x2	1	Surface	SM4x2
	B	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	8	1	Surface	SGPSM
Scattered MSAA Typed Surface R/W	B	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	4x2	1	Surface	SM4x2
	B	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	8	1	Surface	SGPSM
Atomic Operations					2, 8, 16	1		
DW Untyped Atomic Integer	DW	DW	BTS	BUFFER, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	BUFFER, NULL	8, 16	1	Surface	SMPSM
	DW	DW	SLM	BUFFER	4x2	1	Wrap	SM4x2
	DW	DW	SLM	BUFFER	8, 16	1	Wrap	SMPSM
	DW	DW	A32	BUFFER	4x2	1	GenState	SM4x2
	DW	DW	A32	BUFFER	8, 16	1	GenState	SMPSM
	DW	DW	A64	BUFFER	4x2	1	Canonical	SM4x2
	DW	DW	A64	BUFFER	8	1	Canonical	SM
DW Scaled Untyped Atomic Integer	DW	DW	BTS	STRBUF, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	STRBUF, NULL	8, 16	1	Surface	SMPSM
QW Untyped Atomic Integer	QW	QW	BTS	BUFFER, NULL	4x2	1	Surface	SM4x2
	QW	QW	BTS	BUFFER, NULL	8, 16	1	Surface	SMPSM

Operation	Addr Align	Data Width	Address Model	Surface Type	SIMD Slots	Data Elements	Bounds Check	Execution Mask
	QW	QW	A32	BUFFER	4x2	1	GenState	SM4x2
	QW	QW	A32	BUFFER	8, 16	1	GenState	SMPSM
	QW	QW	A64	BUFFER	4x2	1	Canonical	SM4x2
	QW	QW	A64	BUFFER	8	1	Canonical	SM
QW Scaled Untyped Atomic Integer	QW	QW	BTS	STRBUF, NULL	4x2	1	Surface	SM4x2
	QW	QW	BTS	STRBUF, NULL	8, 16	1	Surface	SMPSM
OW Untyped Atomic Integer	OW	OW	A64	BUFFER	4x2	1	Canonical	SM4x2
	OW	OW	A64	BUFFER	8	1	Canonical	SM
DW Untyped Atomic Float	DW	DW	BTS	BUFFER, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	BUFFER, NULL	8, 16	1	Surface	SMPSM
	DW	DW	SLM	BUFFER	4x2	1	Wrap	SM4x2
	DW	DW	SLM	BUFFER	8, 16	1	Wrap	SMPSM
	DW	DW	A32	BUFFER	4x2	1	GenState	SM4x2
	DW	DW	A32	BUFFER	8, 16	1	GenState	SMPSM
	DW	DW	A64	BUFFER	4x2	1	Canonical	SM4x2
	DW	DW	A64	BUFFER	8	1	Canonical	SM
DW Scaled Untyped Atomic Float	DW	DW	BTS	STRBUF, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	STRBUF, NULL	8, 16	1	Surface	SMPSM
DW Typed Atomic Integer	DW	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	4x2	1	Surface	SM4x2
	DW	DW	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	8	1	Surface	SGPSM
DW Atomic Counter	DW	DW	BTS	Any	4x2	1	Ignored	SM4x2
	DW	DW	BTS	Any	8	1	Ignored	SGPSM, SG
Media Block R/W								
Transpose Read	DW	DW	BTS	2D, NULL	1, 2, 4, 8	{1, 2, 4, 8}	Media	Ignored
Media Block R/W	DW	DW	BTS	2D, NULL	1	Height x Width	Media	Ignored
Others								
Read Surface Info	B	B	BTS	Any	1	1	Ignored	Ignored
Memory Fence	N/A	N/A	N/A	N/A	N/A	N/A	Ignored	Ignored

Scattered and Block Read/Write Messages

This section lists the read and write messages that support scattered and block accesses of the standard untyped data: Byte, Word, DWord, QWord, OWords, and HWords.

Byte Scattered ReadWrite Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
B	B	R/W	BTS	BUFFER, NULL	8, 16	1, 2, 4	(Base + GlobalOffset + Offset{Slot})→B[Elem]	Surface	SMBLK
B	B	R/W	SLM	BUFFER	8, 16	1, 2, 4	(Base + Offset{Slot})→B[Elem]	Wrap	SMBLK
B	B	R/W	A32	BUFFER	8, 16	1, 2, 4	(Base + Buffer + GlobalOffset + Offset{Slot})→B[Elem]	PTSS	SMBLK
B	B	R/W	A32	BUFFER	8, 16	1, 2, 4	(Base + (Offset{Slot})→B[Elem]	GenState	SMBLK
B	B	R/W	A64	BUFFER	8, 16	1, 2, 4	(0 + Offset{Slot})→B[Elem]	Canonical	SMBLK

This message reads or writes 8 or 16 scattered and possibly misaligned Bytes, Words, or DWords, starting at each Offset. The offsets are byte-aligned. The **Global Offset** is added to each of the specific offsets when the message header is present.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

- Byte aligned buffer accesses in programmable shaders

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER, NULL	SIMD8	{1, 2, 4}	MSD0R_BS	{Opt} MH_BTS_GO	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	BTS	BUFFER, NULL	SIMD8	{1, 2, 4}	MSD0W_BS	{Opt} MH_BTS_GO	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER, NULL	SIMD16	{1, 2, 4}	MSD0R_BS	{Opt} MH_BTS_GO	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	BUFFER, NULL	SIMD16	{1, 2, 4}	MSD0W_BS	{Opt} MH_BTS_GO	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD8	{1, 2, 4}	MSD0R_BS	{Forbidden}	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	{1, 2, 4}	MSD0W_BS	{Forbidden}	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD16	{1, 2, 4}	MSD0R_BS	{Forbidden}	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	{1, 2, 4}	MSD0W_BS	{Forbidden}	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD8	{1, 2, 4}	MSD0R_BS	{Opt} MH_A32_GO	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	{1, 2, 4}	MSD0W_BS	{Opt} MH_A32_GO	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD16	{1, 2, 4}	MSD0R_BS	{Opt} MH_A32_GO	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	{1, 2, 4}	MSD0W_BS	{Opt} MH_A32_GO	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD8	{1, 2, 4}	MSD2R_BS	{Forbidden}	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	{1, 2, 4}	MSD2W_BS	{Forbidden}	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD16	{1, 2, 4}	MSD2R_BS	{Forbidden}	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	{1, 2, 4}	MSD2W_BS	{Forbidden}	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD1R_A6	{Forbidde	MAP64b_SI	{Forbidden}	{1, 1, 1}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
					4_BS	n}	MD8		MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD1W_A64_BS	{Forbidden}	MAP64b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2R_A64_BS	{Forbidden}	MAP64b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2W_A64_BS	{Forbidden}	MAP64b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2, 4}	MSD1R_A64_BS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2, 4}	MSD1W_A64_BS	{Forbidden}	MAP64b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2R_A64_BS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2W_A64_BS	{Forbidden}	MAP64b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}

Programming Note	
Context:	Byte Scattered ReadWrite Messages
While the hardware does check for and optimize for cases where offsets are equal or contiguous, for optimal performance in some of these cases a different message may provide higher performance.	

Programming Note	
Context:	Byte Scattered ReadWrite Messages
For byte scattered read and write the buffer size must be a multiple of 4 bytes.	

Byte Scaled Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
B	B	R/W	SLM	BUFFER	8, 16	1, 2, 4	(Base + Sideband + (Offset{Slot} * (Pitch+1)))→B[Elem]	Wrap	SMBLK
B	B	R/W	A32	BUFFER	8, 16	1, 2, 4	(Base + Buffer + Sideband + (Offset{Slot} * (Pitch+1)))→B[Elem]	GenState	SMBLK
B	B	R/W	A64	BUFFER	8, 16	1, 2, 4	(0 + Buffer + Sideband + (Offset{Slot} * (Pitch+1)))→B[Elem]	Canonical	SMBLK

This message reads or writes 8 or 16 scattered and possibly misaligned Bytes, Words, or DWords, starting at each Offset. The offsets are scaled by the **Pitch**. The Global Offset is added to each of the specific offsets when the message header is present. The surface format is ignored, data is returned from the buffer to the GRF without format conversion.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	SLM	BUFFER	SIMD8	{1, 2, 4}	MSD2R_BS	{Forbidden}	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	{1, 2, 4}	MSD2W_BS	{Forbidden}	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD16	{1, 2, 4}	MSD2R_BS	{Forbidden}	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	{1, 2, 4}	MSD2W_BS	{Forbidden}	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD8	{1, 2, 4}	MSD2R_BS	MH2_A32	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	{1, 2, 4}	MSD2W_BS	MH2_A32	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A32	BUFFER	SIMD16	{1, 2, 4}	MSD2R_BS	MH2_A32	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	{1, 2, 4}	MSD2W_BS	MH2_A32	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2R_A64_BS	MH2_A64	MAP32b_SIMD8	{Forbidden}	{1, 1, 1} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2W_A64_BS	MH2_A64	MAP32b_SIMD8	{1, 1, 1} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2R_A64_BS	MH2_A64	MAP32b_SIMD16	{Forbidden}	{1, 1, 1} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2W_A64_BS	MH2_A64	MAP32b_SIMD16	{1, 1, 1} MDP_DW_SIMD16	{Forbidden}

Programming Note	
Context:	Byte Scaled Read/Write Messages
For byte scattered read and write the buffer size must be a multiple of 4 bytes.	

DWord Scattered Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	BUFFER, NULL	8, 16	1	Base→DW[GlobalOffset+Offset{Slot}+0]	Surface	SM
DW	DW	R/W	A32	BUFFER	8, 16	1	(Base+Buffer)→DW[GlobalOffset+Offset{Slot}+0]	PTSS	SM
DW	DW	R/W	A64	BUFFER	8, 16	1, 2, 4, 8	(0+Offset{Slot})→DW[Elem]	Canonical	SMBLK

This message reads or writes 8 or 16 scattered DWords starting at each offset. The **Global Offset** is added to each of the specific offsets when it is provided in the message header.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

- SIMD8/16 constant buffer reads where the indices of each pixel are different (read one channel per message)
- SIMD8/16 scratch space reads/writes where the indices are different (read/write one channel per message)
- General purpose DWord scatter/gathering, used by media

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	SIMD8	1	MSD0R_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	1	MSD0W_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	1	MSD0R_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	1	MSD0W_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD8	1	MSD0R_DWS	{Opt} MH_A32_GO	MAP32b_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	1	MSD0W_DWS	{Opt} MH_A32_GO	MAP32b_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD16	1	MSD0R_DWS	{Opt} MH_A32_GO	MAP32b_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	1	MSD0W_DWS	{Opt} MH_A32_GO	MAP32b_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD8	{1, 2, 4, 8}	MSD1R_A64_DWS	{Forbidden}	MAP64b_SIMD8	{Forbidden}	{1, 2, 4, 8} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4, 8}	MSD1W_A64_DWS	{Forbidden}	MAP64b_SIMD8	{1, 2, 4, 8} MDP_DW_SIMD8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD 8	{1, 2, 4, 8}	MSD2R_A64_DWS	{Forbidden}	MAP64b_SIMD8	{Forbidden}	{1, 2, 4, 8} MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	{1, 2, 4, 8}	MSD2W_A64_DWS	{Forbidden}	MAP64b_SIMD8	{1, 2, 4, 8} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD 16	{1, 2, 4}	MSD1R_A64_DWS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 2, 4} MDP_DW_SIMD16
W	A64	BUFFER	SIMD 16	{1, 2, 4}	MSD1W_A64_DWS	{Forbidden}	MAP64b_SIMD16	{1, 2, 4} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD 16	{1, 2, 4}	MSD2R_A64_DWS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 2, 4} MDP_DW_SIMD16
W	A64	BUFFER	SIMD 16	{1, 2, 4}	MSD2W_A64_DWS	{Forbidden}	MAP64b_SIMD16	{1, 2, 4} MDP_DW_SIMD16	{Forbidden}
R	CC BTS	BUFFER,NULL	SIMD 8	1	MSD_CC_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD8	{Forbidden}	MDP_DW_SIMD8
R	CC BTS	BUFFER,NULL	SIMD 16	1	MSD_CC_DWS	{Opt} MH_BTS_GO	MAP32b_SIMD16	{Forbidden}	MDP_DW_SIMD16

Programming Note	
Context:	DWord Scattered Read/Write Messages
For BTS and A32 address models, the offsets in the address payload must in the range 0 - 3FFFFFFFh. If the upper 2 bits are non-zero, the result is undefined.	

DWord Scaled Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	A64	BUFFER	8, 16	1, 2, 4, 8	(0+Buffer+Sideband+(Offset{Slot}*(Pitch+1)))→DW[Elem]	Canonical	SMBLK

This message reads or writes 8 or 16 scattered DWords starting at each offset. The offsets are scaled by the **Pitch**.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	{1, 2, 4, 8}	MSD2R_A64_DWS	MH2_A64	MAP32b_SIMD8	{Forbidden}	{1, 2, 4, 8} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4, 8}	MSD2W_A64_DWS	MH2_A64	MAP32b_SIMD8	{1, 2, 4, 8} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2R_A64_DWS	MH2_A64	MAP32b_SIMD16	{Forbidden}	{1, 2, 4} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2, 4}	MSD2W_A64_DWS	MH2_A64	MAP32b_SIMD16	{1, 2, 4} MDP_DW_SIMD16	{Forbidden}

QWord Scattered Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
QW	QW	R/W	A64	BUFFER	8, 16	1, 2, 4	(0+Offset{Slot})→QW[Elem]	Canonical	SMBLK

This message takes a set of QWord-aligned offsets, and reads or writes SIMD8 scattered QWords starting at each offset.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

SIMD8/16 reads where the indices differ (read one channel per message).

SIMD8/16 writes where the indices differ (write one channel per message).

General purpose QWord scatter/gathering, used by media.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD1R_A64_QWS	{Forbidden}	MAP64b_SIMD8	{Forbidden}	{1, 2, 4} MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD1W_A64_QWS	{Forbidden}	MAP64b_SIMD8	{1, 2, 4} MDP_QW_SIMD8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2R_A64_QWS	{Forbidden}	MAP64b_SIMD8	{Forbidden}	{1, 2, 4} MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2W_A64_QWS	{Forbidden}	MAP64b_SIMD8	{1, 2, 4} MDP_QW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2}	MSD1R_A64_QWS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 2} MDP_QW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2}	MSD1W_A64_QWS	{Forbidden}	MAP64b_SIMD16	{1, 2} MDP_QW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD16	{1, 2}	MSD2R_A64_QWS	{Forbidden}	MAP64b_SIMD16	{Forbidden}	{1, 2} MDP_QW_SIMD16
W	A64	BUFFER	SIMD16	{1, 2}	MSD2W_A64_QWS	{Forbidden}	MAP64b_SIMD16	{1, 2} MDP_QW_SIMD16	{Forbidden}

QWord Scaled Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
QW	QW	R/W	A64	BUFFER	8, 16	1, 2, 4	(0+Buffer+Sideband+(Offset{Slot}*(Pitch+1)))→QW[Elem]	Canonical	SMBLK

This message takes a set of QWord-aligned offsets, and reads or writes SIMD8 scattered QWords starting at each offset. The offsets are scaled by the **Pitch**.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2R_A64_QWS	MH2_A64	MAP32b_SIMD8	{Forbidden}	{1, 2, 4} MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	{1, 2, 4}	MSD2W_A64_QWS	MH2_A64	MAP32b_SIMD8	{1, 2, 4} MDP_QW_SIMD8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD 16	{1, 2}	MSD2R_A64_QWS	MH2_A64	MAP32b_SIMD16	{Forbidden}	{1, 2} MDP_QW_SIMD16
W	A64	BUFFER	SIMD 16	{1, 2}	MSD2W_A64_QWS	MH2_A64	MAP32b_SIMD16	{1, 2} MDP_QW_SIMD16	{Forbidden}

OWord Block Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
OW	OW	R/W	BTS	BUFFER,NULL	1	1, 2, 4, 8	Base→OW[GlobalOffset+Elem]	Surface	BLK
OW	OW	R/W	A32	BUFFER	1	1, 2, 4, 8	(Base+Buffer)→OW[GlobalOffset+Elem]	PTSS	BLK
OW	OW	R/W	A64	BUFFER	1	1, 2, 4, 8	(Base+BlockOffset0)→OW[Elem]	Canonical	BLK

This message takes one offset (Global Offset), and reads or writes 1, 2, 4, or 8 contiguous OWords starting at that offset.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

The execution mask bits may disable accesses on the corresponding blocks. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

- Constant buffer reads of a single constant or multiple contiguous constants.
- Scratch space reads/writes where the index for each pixel/vertex is the same.
- Block constant reads, scratch memory reads/writes for media.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD0R_OWB	MH_BTS_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD0W_OWB	MH_BTS_GO	{Forbidden}	{1, 1, 2, 4} MDP_OW_B	{Forbidden}
R	A32	BUFFER	1	{1, 2, 4, 8}	MSD0R_OWB	MH_A32_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
W	A32	BUFFER	1	{1, 2, 4, 8}	MSD0W_OWB	MH_A32_GO	{Forbidden}	{1, 1, 2, 4} MDP_OW_B	{Forbidden}
R	A64	BUFFER	1	{1, 2, 4, 8}	MSD1R_A64_OWB	MH_A64_OWB	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
W	A64	BUFFER	1	{1, 2, 4, 8}	MSD1W_A64_OWB	MH_A64_OWB	{Forbidden}	{1, 1, 2, 4} MDP_OW_B	{Forbidden}
R	CC BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD_CC_OWB	MH_BTS_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B

For the one-constant case, either the high or low half of the payload register is used depending on the half selected in Block Size, and the other half of the payload register is ignored.

Programming Note	
Context:	OWord Block Read/Write Messages
For the BTS and A32 address models, the offsets in the address payload must be in the range 0 - 0FFFFFFFh. If the upper 4 bits are non-zero, the result is undefined.	

OWord Aligned Block Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	OW	R	BTS	BUFFER, NULL	1	1, 2, 4, 8	(Base+GlobalOffset)→OW[Elem]	Surface	Ignored
DW	OW	R	A32	BUFFER	1	1, 2, 4, 8	(Base+Buffer+GlobalOffset)→OW[Elem]	PTSS	Ignored
DW	OW	R	A64	BUFFER	1	1, 2, 4, 8	(0+BlockOffset0)→OW[Elem]	Canonical	Ignored

Functional Description
This message takes one DWord-aligned offset (Global Offset), and reads 1, 2, 4, or 8 contiguous OWords starting at that offset.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

This message is identical to the OWord Block Read/Write message except the offset alignment. The Global Offset is specified in bytes, and must be aligned to the Addr Align entry in table.

The execution mask is ignored for this message. Out-of-bounds accesses are dropped or return zero.

Applications:

Block accesses directly using a byte offset instead of an Oword index.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD0R_OWAB	MH_BTS_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
R	A32	BUFFER	1	{1, 2, 4, 8}	MSD0R_OWAB	MH_A32_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
R	A64	BUFFER	1	{1, 2, 4, 8}	MSD1R_A64_OWAB	MH_A64_OWB	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
R	CC BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD_CC_OWAB	MH_BTS_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B
R	SC BTS	BUFFER,NULL	1	{1, 2, 4, 8}	MSD_SC_OWAB	MH_BTS_GO	{Forbidden}	{Forbidden}	{1, 1, 2, 4} MDP_OW_B

Programming Note
For read/write cache, only the read path supports the unaligned OWord Block access.

OWord Dual Block Read/Write Messages

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
OW	OW	R / W	BTS	BUFFER,NULL	2	1, 4	Base→OW[GlobalOffset+Offset{0, 1}+Elem]	Surface	DBLK
OW	OW	R / W	A32	BUFFER	2	1, 4	(Base+Buffer)→OW[GlobalOffset+Offset{0, 1}+Elem]	PTSS	DBLK
OW	OW	R / W	A64	BUFFER	2	1, 4	(0+BlockOffset{0, 1})→OW[Elem]	Canonical	DBLK

This message takes two offsets, and reads or writes 1 or 4 contiguous OWords starting at each offset. The Global Offset is added to each of the specific offsets.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

The execution mask bits may disable accesses on the corresponding blocks. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

- SIMD4x2 constant buffer reads where the indices of each vertex/pixel differ (if two indices are the same, hardware optimizes the cache accesses and does only one cache access).
- SIMD4x2 scratch space reads/writes where the indices differ.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER, NULL	2	{1, 4}	MSD0R_OWDB	{Opt} MH_BTS_GO	MAP32b_SIMD4x2	{Forbidden}	{1, 4} MDP_OW_DB
W	BTS	BUFFER, NULL	2	{1, 4}	MSD0W_OWDB	{Opt} MH_BTS_GO	MAP32b_SIMD4x2	{1, 4} MDP_OW_DB	{Forbidden}
R	A32	BUFFER	2	{1, 4}	MSD0R_OWDB	{Opt} MH_A32_GO	MAP32b_SIMD4x2	{Forbidden}	{1, 4} MDP_OW_DB
W	A32	BUFFER	2	{1, 4}	MSD0W_OWDB	{Opt} MH_A32_GO	MAP32b_SIMD4x2	{1, 4} MDP_OW_DB	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	2	{1, 4}	MSD1R_A64_OWDB	MH_A64_OWDB	{Forbidden}	{Forbidden}	{1, 4} MDP_OW_DB
W	A64	BUFFER	2	{1, 4}	MSD1W_A64_OWDB	MH_A64_OWDB	{Forbidden}	{1, 4} MDP_OW_DB	{Forbidden}
R	CC BTS	BUFFER, NULL	2	{1, 4}	MSD_CC__OWDB	{Opt} MH_BTS_GO	MAP32b_SIMD4x2	{Forbidden}	{1, 4} MDP_OW_DB

Programming Note	
Context:	OWord Dual Block Read/Write Messages
For BTS and A32 address models, the offsets in the address payload must in the range 0 - 0FFFFFFFh. If the upper 4 bits are non-zero, the result is undefined.	

HWord Scratch Block Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
HW	HW	R/W	A32	BUFFER	1	1, 2, 4, 8	(Base+Buffer)→HW[SB_Offset+Elem]	PTSS	BLKCM

This message performs a read or write operation of between 1 and 8 registers to an HWord-aligned offset.

The execution mask bits may disable accesses on the corresponding blocks. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

Applications:

Scratch space reads/writes for register spill/fill operations.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A32	BUFFER	1	{1, 2, 4, 8}	MSD0R_SB	MH_A32_HWB	{Forbidden}	{Forbidden}	{1, 2, 4, 8} MDP_HW_B
W	A32	BUFFER	1	{1, 2, 4, 8}	MSD0W_SB	MH_A32_HWB	{Forbidden}	{1, 2, 4, 8} MDP_HW_B	{Forbidden}

Channel Mode Interpretation
Two modes of channel-enable interpretation are provided: • DWord, which support a SIMD8 or SIMD16 DWord channel-serial view of a register, and • OWord, which supports a SIMD4x2 view of a register.

Programming Note	
Context:	HWord Block ReadWrite Messages
For the MSD0R_SB and MSD0W_SB messages, the HWord offset into the A32 memory is provided in the message descriptor and communicated to the data port on the Sideband. This allows a single instruction read|write block operation in a single source instruction. The message descriptor provides 12 bits for the HWord offset, allowing addressing of 4K HWord locations (128KB) based from from the A32 general state base address.	

Programming Note
Although Scratch Block messages are always used only inside a thread's local scratch space, they are assumed to be coherent memory accesses (MDC_STATELESS A32_A64).

HWord Aligned Block Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	HW	R/W	A64	BUFFER	1	1, 2, 4, 8	(0+BlockOffset0)→HW[Elem]	Canonical	BLKCM

Functional Description
This message takes one DWord-aligned offset (Global Offset), and reads 1, 2, 4, or 8 contiguous HWords starting at that offset.

A BTS surface is interpreted as a RAW BUFFER regardless of the surface format. The pitch is 1 byte and data is returned from the buffer to the GRF without format conversion.

This message is identical to the HWord Block message except the offset alignment. The Global Offset is specified in bytes, and must be aligned to the Addr Align entry in the table.

The execution mask is ignored for this message. Out-of-bounds accesses are dropped or return zero.

Applications:

Block accesses directly using a byte offset instead of an Hword index.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	1	{1, 2, 4, 8}	MSD1R_A64_HWB	MH_A64_HWB	{Forbidden}	{Forbidden}	{1, 2, 4, 8} MDP_HW_B
W	A64	BUFFER	1	{1, 2, 4, 8}	MSD1W_A64_HWB	MH_A64_HWB	{Forbidden}	{1, 1, 2, 4} MDP_HW_B	{Forbidden}

Channel Mode Interpretation
Two modes of channel-enable interpretation are provided: • DWord, which support a SIMD8 or SIMD16 DWord channel-serial view of a register, and • OWord, which supports a SIMD4x2 view of a register.

Programming Note
Although Scratch Block messages are only used inside a thread's local scratch space, they are assumed to be coherent memory accesses (MDC_STATELESS A32_A64).

Surface ReadWrite Messages

The surface read and write messages allow direct read/write accesses to untyped and typed surfaces.

Untyped surfaces are either SURFTYPE_BUFFER (format RAW, pitch 1) or SURFTYPE_STRBUF (format RAW, pitch from the surface state). SLM and Stateless messages are always untyped BUFFER accesses.

Typed surfaces are 1D, 2D, 3D, CUBE, or BUFFER types. The surface state specifies all the parameters.There are some limitations on what typed data surface formats are supported by data ports; see the specific messages and Surfaces Types for more details.

Untyped Surface ReadWrite Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	OW	R/W	BTS	BUFFER,NULL	4x2	1	(Base+U{Slot})→DW[Chan]	Surface	SM4x2
DW	DW	R/W	BTS	BUFFER,NULL	8, 16	{Chan 1-4}	(Base+U{Slot})→DW[Chan]	Surface	SMPSM
DW	OW	R/W	SLM	BUFFER	4x2	1	(Base+U{Slot})→DW[Chan]	Wrap	SM4x2
DW	DW	R/W	SLM	BUFFER	8, 16	{Chan 1-4}	(Base+U{Slot})→DW[Chan]	Wrap	SMPSM

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	OW	R/W	A32	BUFFER	4x2	1	(Base+Buffer+U{Slot})→DW[Chan]	GenState	SM4x2
DW	DW	R/W	A32	BUFFER	8, 16	{Chan 1-4}	(Base+Buffer+U{Slot})→DW[Chan]	GenState	SMPSM
DW	OW	R/W	A64	BUFFER	4x2	1	(0+U{Slot})→DW[Chan]	Canonical	SM4x2
DW	DW	R/W	A64	BUFFER	8, 16	{Chan 1-4}	(0+U{Slot})→DW[Chan]	Canonical	SM

This message allows direct read/write accesses to untyped surfaces using 8 or 16 offsets. Up to 4 DWords are accessed beginning at the byte addresses determined. These 4 DWords correspond to the red, green, blue, and alpha channels, in that order and with red mapping to the lowest order DWord. If the U offset is not DWord-aligned, the results are undefined.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_US	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	BTS	BUFFER,NULL	SIMD4x2	1	MSD1W_US	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD8	{Chan 1-4}	MSD1R_US	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	{Chan 1-4}	MSD1W_US	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	{Chan 1-4}	MSD1R_US	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	{Chan 1-4}	MSD1W_US	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD4x2	1	MSD1R_US	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	SLM	BUFFER	SIMD4x2	1	MSD1W_US	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	SLM	BUFFER	SIMD4x2	1	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	SLM	BUFFER	SIMD4x2	1	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	SLM	BUFFER	SIMD8	{Chan 1-4}	MSD1R_US	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	{Chan 1-4}	MSD1W_US	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD8	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD16	{Chan 1-4}	MSD1R_US	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	{Chan 1-4}	MSD1W_US	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD16	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD4x2	1	MSD1R_US	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A32	BUFFER	SIMD4x2	1	MSD1W_US	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	A32	BUFFER	SIMD4x2	1	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A32	BUFFER	SIMD4x2	1	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A32	BUFFER	SIMD8	{Chan 1-4}	MSD1R_US	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	{Chan 1-4}	MSD1W_US	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD8	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD16	{Chan 1-4}	MSD1R_US	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	{Chan 1-4}	MSD1W_US	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD16	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD1R_A64_US	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A64	BUFFER	SIMD4x2	1	MSD1W_A64_US	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD2R_A64_US	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A64	BUFFER	SIMD4x2	1	MSD2W_A64_US	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	A64	BUFFER	SIMD8	{Chan 1-4}	MSD1R_A64_US	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{Chan 1-4}	MSD1W_A64_US	{Forbidden}	MAP64b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	{Chan 1-4}	MSD2R_A64_US	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	{Chan 1-4}	MSD2W_A64_US	{Forbidden}	MAP64b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD16	{Chan 1-4}	MSD1R_A64_US	{Forbidden}	MAP64b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{Chan 1-4}	MSD1W_A64_US	{Forbidden}	MAP64b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD16	{Chan 1-4}	MSD2R_A64_US	{Forbidden}	MAP64b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	A64	BUFFER	SIMD16	{Chan 1-4}	MSD2W_A64_US	{Forbidden}	MAP64b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}

The number of message registers in the write data payload is determined by the number of channel mask bits that are enabled.

The **Channel Mask** in the message descriptor identifies whether the corresponding channel access is disabled. For BTS surfaces, the surface state **Shader Channel Select** fields identify whether the channels are swizzled for the access.

Scaled Untyped Surface Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	OW	R/W	BTS	STRBUF,NULL	4x2	1	(Base+(U{Slot}*Pitch)+V{Slot})→DW[Chan]	Surface	SM4x2
DW	DW	R/W	BTS	STRBUF,NULL	8, 16	{Chan 1-4}	(Base+(U{Slot}*Pitch)+V{Slot})→DW[Chan]	Surface	SMPSM
DW	OW	R/W	SLM	BUFFER	4x2	1	(Base+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	Wrap	SM4x2
DW	DW	R/W	SLM	BUFFER	8, 16	{Chan 1-4}	(Base+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	Wrap	SM
DW	OW	R/W	A32	BUFFER	4x2	1	(Base+Buffer+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	GenState	SM4x2
DW	DW	R/W	A32	BUFFER	8, 16	{Chan 1-4}	(Base+Buffer+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	GenState	SMPSM

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	OW	R/W	A64	BUFFER	4x2	1	(0+Buffer+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	Canonical	SM4x2
DW	DW	R/W	A64	BUFFER	8, 16	{Chan 1-4}	(0+Buffer+Sideband+(U{Slot}*(Pitch+1)))→DW[Chan]	Canonical	SM

This message allows scaled read/write accesses to untyped scaled surfaces using 8 or 16 offsets. Up to 4 DWords are accessed beginning at the byte address determined by the U offset and the scaling pitch. These 4 DWords correspond to the red, green, blue, and alpha channels, in that order and with red mapping to the lowest order DWord. After scaling calculation, the offset must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	STRBUF,NULL	SIMD 4x2	1	MSD1R_US	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	BTS	STRBUF,NULL	SIMD 4x2	1	MSD1W_US	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD 8	{Chan 1-4}	MSD1R_US	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD 8	{Chan 1-4}	MSD1R_US	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	BTS	STRBUF,NULL	SIMD 16	{Chan 1-4}	MSD1W_US	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	BTS	STRBUF,NULL	SIMD 16	{Chan 1-4}	MSD1W_US	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD 4x2	1	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	SLM	BUFFER	SIMD 4x2	1	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	SLM	BUFFER	SIMD 8	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	SLM	BUFFER	SIMD 8	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD 16	{Chan 1-4}	MSD2R_US	{Forbidden}	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	SLM	BUFFER	SIMD 16	{Chan 1-4}	MSD2W_US	{Forbidden}	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD 4x2	1	MSD2R_US	MH2_A32	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A32	BUFFER	SIMD 4x2	1	MSD2W_US	MH2_A32	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	A32	BUFFER	SIMD 8	{Chan 1-4}	MSD2R_US	MH2_A32_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A32	BUFFER	SIMD 8	{Chan 1-4}	MSD2W_US	MH2_A32_PSM	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD 16	{Chan 1-4}	MSD2R_US	MH2_A32_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16
W	A32	BUFFER	SIMD 16	{Chan 1-4}	MSD2W_US	MH2_A32_PSM	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD2R_A64_US	MH2_A64	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	A64	BUFFER	SIMD 4x2	1	MSD2W_A64_US	MH2_A64	MAP32b_USU_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	A64	BUFFER	SIMD 8	{Chan 1-4}	MSD2R_A64_US	MH2_A64	MAP32b_USU_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	{Chan 1-4}	MSD2W_A64_US	MH2_A64	MAP32b_USU_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD 16	{Chan 1-4}	MSD2R_A64_US	MH2_A64	MAP32b_USU_SIMD16	{Forbidden}	{Chan1-4} MDP_DW_SIMD16

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	A64	BUFFER	SIMD 16	{Chan 1-4}	MSD2W_A64_US	MH2_A64	MAP32b_USU_SIMD16	{Chan1-4} MDP_DW_SIMD16	{Forbidden}

Typed Surface Read/Write Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
B	DW	R/W	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	4x2	1	(Surface[U, V, R, LOD])→DW[Chan]	Surface	SM4x2
B	DW	R/W	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	8	1	(Surface[U, V, R, LOD])→DW[Chan]	Surface	SGPSM

This message allows direct read/write accesses to typed surfaces using 8 offsets. Up to 4 channels are accessed beginning at the byte address determined: red, green, blue, and alpha channels, in that order.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD 4x2	1	MSD1R_TS	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	{Forbidden}	MDP_DW_SIMD 4x2
W	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD 4x2	1	MSD1W_TS	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	MDP_DW_SIMD 4x2	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD 8	{Chan 1-4}	MSD1R_TS	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD 8

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD8	{Chan 1-4}	MSD1W_TS	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD8	1	MSD1R_TS	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	{Forbidden}	MDP_TileW_SIMD8
W	BTS	1D, 2D, 3D, CUBE, BUFFER, NULL	SIMD8	1	MSD1W_TS	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	MDP_TileW_SIMD8	{Forbidden}

The number of message registers in the write data payload is determined by the number of channel mask bits that are enabled.

The **Channel Mask** in the message descriptor identifies whether the corresponding channel access is disabled. The surface state **Shader Channel Select** fields identify whether the channels are swizzled for the access.

See Surface Formats for the description of how the data is formatted for Typed Surface read and write operations.

MSAA Typed Surface ReadWrite Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
B	DW	R/W	BTS	2D, NULL	4x2	1	(Surface[U, V, R, LOD, MSAA])→DW[Chan]	Surface	SM4x2
B	DW	R/W	BTS	2D, NULL	8	1	(Surface[U, V, R, LOD, MSAA])→DW[Chan]	Surface	SGPSM

This message allows direct read/write accesses of multi-sample typed surfaces using 8 offsets. Up to 4 channels are accessed beginning at the byte address determined: red, green, blue, and alpha channels, in that order.

The 2D surface can be either a normal 2D surface, a 2D Array surface, or a 2D mip-mapped surface. Sample planes are laid out sequentially for each array element or mip-mapped surface.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. If the MSAA sample number is larger than **Number of Multisamples** in the surface state, the access is out of bounds.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	2D, NULL	SIMD 4x2	1	MSD1R_TS	{Opt} MH_IGNORE	MAP32b_MSAA_TS_SIMD4x2	{Forbidden}	MDP_DW_SIMD4x2
W	BTS	2D, NULL	SIMD 4x2	1	MSD1W_TS	{Opt} MH_IGNORE	MAP32b_MSAA_TS_SIMD4x2	MDP_DW_SIMD4x2	{Forbidden}
R	BTS	2D, NULL	SIMD 8	{Chan 1-4}	MSD1R_TS	{Opt} MH1_BTS_PSM	MAP32b_MSAA_TS_SIMD8	{Forbidden}	{Chan1-4} MDP_DW_SIMD8
W	BTS	2D, NULL	SIMD 8	{Chan 1-4}	MSD1W_TS	{Opt} MH1_BTS_PSM	MAP32b_MSAA_TS_SIMD8	{Chan1-4} MDP_DW_SIMD8	{Forbidden}

The number of message registers in the write data payload is determined by the number of channel mask bits that are enabled.

The **Channel Mask** in the message descriptor identifies whether the corresponding channel access is disabled. The surface state **Shader Channel Select** fields identify whether the channels are swizzled for the access.

See Surface Formats for the description of how the data is formatted for Typed Surface read and write operations.

Programming Note	
Context:	MSAA Typed Surface ReadWrite Messages
This message only supports PLANAR multi-sample surfaces (surface state **Multisample Surface Storage Format** MSS), without a color control auxillary surface (surface state **Auxillary Surface Mode** NONE) and without color compression (surface state **Render Target Compression Enable** FALSE).	

Atomic Operation Messages

Atomic operation messages cause atomic read-modify-write operations on the destination locations addressed. Atomic operations guarantee that no read or write to the same memory location from this thread or any other thread can occur between the read and the write. There is no guarantee on the write ordering of the individual operations in one SIMD message.

Out-of-bounds atomic accesses are dropped without writing data, and return zero for read data. Atomic accesses to pages in Tiled Resources Translation Tables are required to be IA-coherent. NULL Tiled Resources pages are handled as out-of-bounds accesses.

Untyped atomic messages use the RAW format, with surface type BUFFER or STRBUF, and perform no type conversion. These messages use the U address parameter, which specifies the byte offset, which

must be aligned on the data width (multiple of 8 for QWord, a multiple of 4 for DWord, or a multiple of 2 for Word).

Typed atomic messages use surface types SURFTYPE_1D, 2D, 3D, or BUFFER.

For atomic operations, each shader channel select in the surface state must be set to the same surface channel (R = SCS_RED, G = SCS_GREEN, B = SCS_BLUE, A = SCS_ALPHA). Only the red surface channel is used (the first DWord).

The MDC_AOP and MDC_FOP are specified in the Atomic Operation Message Descriptor Control Fields section.

The new value of the destination is computed based on the old value of the destination, and up to two additional sources included in the message (src0 and src1). The sources used by the specific operations are described in the **MDC_AOP** and **MDC_FOP** tables. The AOP operations are subdivided into unary, binary, and trinary operand groupings. Source data payload are only sent for the binary and trinary operations.

The double-width AOP operation CMPWR_2W, also known as CMPRW8B on DWords and CMPWR16B on QWords, require the read-modify-write address offsets to be naturally aligned (QWord and OWord respectively).

The **MDC_RDC** message-specific control field controls whether a value is returned by the atomic message. The value returned depends on the message specific operation (see **MDC_AOP** and **MDC_FOP**).

DWord Untyped Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	BUFFER,NULL	4x2	1	(Base+U{Slot})→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	BUFFER,NULL	8, 16	1	(Base+U{Slot})→DW[0]	Surface	SMPSM
DW	DW	R/W	SLM	BUFFER	4x2	1	(Base+U{Slot})→DW[0]	Wrap	SM4x2
DW	DW	R/W	SLM	BUFFER	8, 16	1	(Base+U{Slot})→DW[0]	Wrap	SMPSM
DW	DW	R/W	A32	BUFFER	4x2	1	(Base+Buffer+U{Slot})→DW[0]	GenState	SM4x2
DW	DW	R/W	A32	BUFFER	8, 16	1	(Base+Buffer+U{Slot})→DW[0]	GenState	SMPSM
DW	DW	R/W	A64	BUFFER	4x2	1	(0+U{Slot})→DW[0]	Canonical	SM4x2
DW	DW	R/W	A64	BUFFER	8	1	(0+U{Slot})→DW[0]	Canonical	SM

This message performs atomic integer operations on untyped surfaces using 8 or 16 offsets. One DWord is accessed beginning at the byte address determined. Each U offset must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_DWAI1_4x2	{Opt}MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	BTS	BUFFER,NULL	SIMD4x2	1	MSD1W_DWAI1_4x2	{Opt}MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	{Forbidden}
R	BTS	BUFFER,NULL	SIMD8	1	MSD1R_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	1	MSD1W_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	1	MSD1R_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	1	MSD1W_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Forbidden}
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_DWAI2_4x2	{Opt}MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	BUFFER,NULL	SIMD4x2	1	MSD1W_DWAI2_4x2	{Opt}MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	BUFFER,NULL	SIMD8	1	MSD1R_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	1	MSD1W_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	1	MSD1R_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD15	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	1	MSD1W_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD15	{Forbidden}
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_DWAI3_4x2	{Opt}MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	BUFFER,NULL	SIMD 4x2	1	MSD1W_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD 8	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD 8	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD 16	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD 16	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}
R	SLM	BUFFER	SIMD 4x2	1	MSD1R_DWAI1_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	SLM	BUFFER	SIMD 4x2	1	MSD1W_DWAI1_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	{Forbidden}
R	SLM	BUFFER	SIMD 8	1	MSD1R_DWAI1	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	SLM	BUFFER	SIMD 8	1	MSD1W_DWAI1	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Forbidden}
R	SLM	BUFFER	SIMD 16	1	MSD1R_DWAI1	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	SLM	BUFFER	SIMD 16	1	MSD1W_DWAI1	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Forbidden}
R	SLM	BUFFER	SIMD 4x2	1	MSD1R_DWAI2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	SLM	BUFFER	SIMD 4x2	1	MSD1W_DWAI2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	SLM	BUFFER	SIMD8	1	MSD1R_DWAI2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	1	MSD1W_DWAI2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD16	1	MSD1R_DWAI2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	1	MSD1W_DWAI2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD4x2	1	MSD1R_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	SLM	BUFFER	SIMD4x2	1	MSD1W_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	SLM	BUFFER	SIMD8	1	MSD1R_DWAI3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	SLM	BUFFER	SIMD8	1	MSD1W_DWAI3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	SLM	BUFFER	SIMD16	1	MSD1R_DWAI3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	SLM	BUFFER	SIMD16	1	MSD1W_DWAI3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}
R	A32	BUFFER	SIMD4x2	1	MSD1R_DWAI1_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	A32	BUFFER	SIMD4x2	1	MSD1W_DWAI1_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	{Forbidden}	{Forbidden}
R	A32	BUFFER	SIMD8	1	MSD1R_DWAI1	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	1	MSD1W_DWAI1	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A32	BUFFER	SIMD16	1	MSD1R_DWAI1	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	1	MSD1W_DWAI1	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	{Forbidden}	{Forbidden}
R	A32	BUFFER	SIMD4x2	1	MSD1R_DWAI2_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	A32	BUFFER	SIMD4x2	1	MSD1W_DWAI2_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	A32	BUFFER	SIMD8	1	MSD1R_DWAI2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	1	MSD1W_DWAI2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD16	1	MSD1R_DWAI2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	1	MSD1W_DWAI2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD4x2	1	MSD1R_DWAI3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	A32	BUFFER	SIMD4x2	1	MSD1W_DWAI3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	A32	BUFFER	SIMD8	1	MSD1R_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	A32	BUFFER	SIMD8	1	MSD1W_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	A32	BUFFER	SIMD16	1	MSD1R_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	A32	BUFFER	SIMD16	1	MSD1W_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD1R_A64_DWAI1_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_DWAI1_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	{Forbidden}
R	A64	BUFFER	SIMD 8	1	MSD1R_A64_DWAI1	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	1	MSD1W_A64_DWAI1	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_DWAI2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_DWAI2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	A64	BUFFER	SIMD 8	1	MSD1R_A64_DWAI2	{Forbidden}	MAP64b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	1	MSD1W_A64_DWAI2	{Forbidden}	MAP64b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_DWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_DWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	A64	BUFFER	SIMD 8	1	MSD1R_A64_DWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	1	MSD1W_A64_DWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}

Programming Note	
Context:	DWord Untyped Atomic Integer Messages
AOP imax/imin assume operands are signed 32-bit integers, umax/umin assume operands are unsigned integers. All other operations treat all values as 32-bit unsigned integers. Add and subtract operations wrap without any special indication.	

DWord Scaled Untyped Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	STRBUF,NULL	4x2	1	(Base+(U{Slot}*Pitch)+V{Slot})→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	STRBUF,NULL	8, 16	1	(Base+(U{Slot}*Pitch)+V{Slot})→DW[0]	Surface	SMPSM

This message performs atomic integer operations on untyped surfaces using 8 or 16 offsets. One DWord is accessed beginning at the byte address determined. Each U offset is scaled by Pitch. Both the Surface Pitch and V offset must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	STRBUF,NULL	SIMD4x2	1	MSD1R_DWAI1_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	BTS	STRBUF,NULL	SIMD4x2	1	MSD1W_DWAI1_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	{Forbidden}	{Forbidden}
R	BTS	STRBUF,NULL	SIMD8	1	MSD1R_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD8	1	MSD1W_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	{Forbidden}	{Forbidden}
R	BTS	STRBUF,NULL	SIMD16	1	MSD1R_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD16	{Forbidden}	MDP_DW_SIMD16
W	BTS	STRBUF,NULL	SIMD16	1	MSD1W_DWAI1	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD16	{Forbidden}	{Forbidden}
R	BTS	STRBUF,NULL	SIMD4x2	1	MSD1R_DWAI2_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	STRBUF,NULL	SIMD4x2	1	MSD1W_DWAI2_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	STRBUF,NULL	SIMD8	1	MSD1R_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD8	1	MSD1W_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	STRBUF,NULL	SIMD16	1	MSD1R_DWAI2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	STRBUF,NULL	SIMD16	1	MSD1W_DWAI2	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	BTS	STRBUF,NULL	SIMD4x2	1	MSD1R_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	BTS	STRBUF,NULL	SIMD4x2	1	MSD1W_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD8	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD8	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD16	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	BTS	STRBUF,NULL	SIMD16	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_DW2	{Forbidden}

QWord Untyped Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
QW	QW	R/W	BTS	BUFFER,NULL	4x2	1	(Base+U{Slot})→QW[0]	Surface	SM4x2
QW	QW	R/W	BTS	BUFFER,NULL	8, 16	1	(Base+U{Slot})→QW[0]	Surface	SMPSM
QW	QW	R/W	A32	BUFFER	4x2	1	(Base+Buffer+U{Slot})→QW[0]	GenState	SM4x2
QW	QW	R/W	A32	BUFFER	8, 16	1	(Base+Buffer+U{Slot})→QW[0]	GenState	SMPSM
QW	QW	R/W	A64	BUFFER	4x2	1	(0+U{Slot})→QW[0]	Canonical	SM4x2
QW	QW	R/W	A64	BUFFER	8	1	(0+U{Slot})→QW[0]	Canonical	SM

This message performs QWord atomic integer operations on untyped surfaces using 8 or 16 offsets. The surface format is RAW and the surface type is SURFTYPE_BUFFER. One QWord is accessed beginning at the byte address determined. Each U offset must be QWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	SIMD 4x2	1	MSD1R_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_QW2	MDP_AOP4x2_QW1
W	BTS	BUFFER,NULL	SIMD 4x2	1	MSD1W_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_QW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD 8	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_QW2	MDP_AOP8_QW1
W	BTS	BUFFER,NULL	SIMD 8	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_QW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD 16	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_QW2	MDP_AOP16_QW1
W	BTS	BUFFER,NULL	SIMD 16	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_QW2	{Forbidden}
R	A32	BUFFER	SIMD 4x2	1	MSD1R_DWAI3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_QW2	MDP_AOP4x2_QW1
W	A32	BUFFER	SIMD 4x2	1	MSD1W_DWAI3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_QW2	{Forbidden}
R	A32	BUFFER	SIMD 8	1	MSD1R_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_QW2	MDP_AOP8_QW1
W	A32	BUFFER	SIMD 8	1	MSD1W_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_QW2	{Forbidden}
R	A32	BUFFER	SIMD 16	1	MSD1R_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_QW2	MDP_AOP16_QW1
W	A32	BUFFER	SIMD 16	1	MSD1W_DWAI3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_QW2	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_QWAI1_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_QW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_QWAI1_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	{Forbidden}	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	1	MSD1R_A64_QWAI1	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	1	MSD1W_A64_QWAI1	{Forbidden}	MAP64b_USU_SIMD8	{Forbidden}	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD1R_A64_QWAI2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_QW1	MDP_AOP4x2_QW1
W	A64	BUFFER	SIMD4x2	1	MSD1W_A64_QWAI2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_QW1	{Forbidden}
R	A64	BUFFER	SIMD8	1	MSD1R_A64_QWAI2	{Forbidden}	MAP64b_USU_SIMD8	MDP_QW_SIMD8	MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	1	MSD1W_A64_QWAI2	{Forbidden}	MAP64b_USU_SIMD8	MDP_QW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD1R_A64_QWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_QW2	MDP_AOP4x2_QW1
W	A64	BUFFER	SIMD4x2	1	MSD1W_A64_QWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_QW2	{Forbidden}
R	A64	BUFFER	SIMD8	1	MSD1R_A64_QWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_QW2	MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	1	MSD1W_A64_QWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_QW2	{Forbidden}
R	A64	BUFFER	SIMD4x2	1	MSD1R_A64_DWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_QW2	MDP_AOP4x2_QW1
W	A64	BUFFER	SIMD4x2	1	MSD1W_A64_DWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_QW2	{Forbidden}
R	A64	BUFFER	SIMD8	1	MSD1R_A64_DWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_QW2	MDP_QW_SIMD8
W	A64	BUFFER	SIMD8	1	MSD1W_A64_DWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_QW2	{Forbidden}

Programming Note	
Context:	QWord Untyped Atomic Integer Messages
AOP imax/imin assume operands are signed 64-bit integers; umax/umin assume operands are unsigned integers. All other operations treat all values as 64-bit unsigned integers. Add and subtract operations wrap without any special indication.	

QWord Scaled Untyped Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
QW	QW	R/W	BTS	STRBUF,NULL	4x2	1	(Base+(U{Slot}*Pitch)+V{Slot})→QW[0]	Surface	SM4x2
QW	QW	R/W	BTS	STRBUF,NULL	8, 16	1	(Base+(U{Slot}*Pitch)+V{Slot})→QW[0]	Surface	SMPSM

This message performs QWord atomic integer operations on untyped surfaces using 8 or 16 offsets. The surface format is RAW and the surface type is SURFTYPE_STRBUF. One QWord is accessed beginning at the byte address determined. Each U offset is scaled by Pitch. Both the Surface Pitch and V must be QWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	STRBUF,NULL	SIMD 4x2	1	MSD1R_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_QW2	MDP_AOP4x2_QW1
W	BTS	STRBUF,NULL	SIMD 4x2	1	MSD1W_DWAI3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_QW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD 8	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_QW2	MDP_AOP16_QW1
W	BTS	STRBUF,NULL	SIMD 8	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_QW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD 16	1	MSD1R_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_QW2	MDP_AOP16_QW1
W	BTS	STRBUF,NULL	SIMD 16	1	MSD1W_DWAI3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_QW2	{Forbidden}

Programming Note	
Context:	QWord Scaled Untyped Atomic Integer Messages
AOP imax/imin assume operands are signed 64-bit integers; umax/umin assume operands are unsigned integers. All other operations treat all values as 64-bit unsigned integers. Add and subtract operations wrap without any special indication.	

OWord Untyped Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
OW	OW	R/W	A64	BUFFER	4x2	1	(Base+U{Slot})→OW[0]	Canonical	SM4x2
OW	OW	R/W	A64	BUFFER	8	1	(Base+U{Slot})→QW[0]	Canonical	SM

This message performs OWord atomic integer CMPWR on untyped surfaces using 8 or 16 offsets. Only A64 stateless addresses are supported. The surface format is RAW and the surface type is SURFTYPE_BUFFER. One QWord is accessed beginning at the byte address determined. Each U offset must be OWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_QWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_OW2	MDP_A64_AOP4x2_OW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_QWAI3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_A64_AOP4x2_OW2	{Forbidden}
R	A64	BUFFER	SIMD 8	1	MSD1R_A64_QWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_OW2	MDP_OW8
W	A64	BUFFER	SIMD 8	1	MSD1W_A64_QWAI3	{Forbidden}	MAP64b_USU_SIMD8	MDP_A64_AOP8_OW2	{Forbidden}

DWord Untyped Atomic Float Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	BUFFER,NULL	4x2	1	(Base+U{Slot})→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	BUFFER,NULL	8, 16	1	(Base+U{Slot})→DW[0]	Surface	SMPSM
DW	DW	R/W	SLM	BUFFER	4x2	1	(Base+U{Slot})→DW[0]	Wrap	SM4x2
DW	DW	R/W	SLM	BUFFER	8, 16	1	(Base+U{Slot})→DW[0]	Wrap	SMPSM
DW	DW	R/W	A32	BUFFER	4x2	1	(Base+Buffer+U{Slot})→DW[0]	GenState	SM4x2
DW	DW	R/W	A32	BUFFER	8, 16	1	(Base+Buffer+U{Slot})→DW[0]	GenState	SMPSM
DW	DW	R/W	A64	BUFFER	4x2	1	(0+U{Slot})→DW[0]	Canonical	SM4x2
DW	DW	R/W	A64	BUFFER	8	1	(0+U{Slot})→DW[0]	Canonical	SM

This message performs atomic float operations on untyped surfaces using 8 or 16 offsets. One DWord is accessed beginning at the byte address determined. Each U offset must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_DWAF2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	BUFFER,NULL	SIMD4x2	1	MSD1W_DWAF2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	BUFFER,NULL	SIMD8	1	MSD1R_DWAF2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	1	MSD1W_DWAF2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	1	MSD1R_DWAF2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	1	MSD1W_DWAF2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	BTS	BUFFER,NULL	SIMD4x2	1	MSD1R_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	BTS	BUFFER,NULL	SIMD4x2	1	MSD1W_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD8	1	MSD1R_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	BUFFER,NULL	SIMD8	1	MSD1W_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	BTS	BUFFER,NULL	SIMD16	1	MSD1R_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	BTS	BUFFER,NULL	SIMD16	1	MSD1W_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	SLM	BUFFER	SIMD 4x2	1	MSD1R_DWAF2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	SLM	BUFFER	SIMD 4x2	1	MSD1W_DWAF2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	SLM	BUFFER	SIMD 8	1	MSD1R_DWAF2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	SLM	BUFFER	SIMD 8	1	MSD1W_DWAF2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	SLM	BUFFER	SIMD 16	1	MSD1R_DWAF2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16
W	SLM	BUFFER	SIMD 16	1	MSD1W_DWAF2	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	SLM	BUFFER	SIMD 4x2	1	MSD1R_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	SLM	BUFFER	SIMD 4x2	1	MSD1W_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	SLM	BUFFER	SIMD 8	1	MSD1R_DWAF3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	SLM	BUFFER	SIMD 8	1	MSD1W_DWAF3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	SLM	BUFFER	SIMD 16	1	MSD1R_DWAF3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	SLM	BUFFER	SIMD 16	1	MSD1W_DWAF3	{Opt} MH1_SLM_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}
R	A32	BUFFER	SIMD 4x2	1	MSD1R_DWAF2_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	A32	BUFFER	SIMD 4x2	1	MSD1W_DWAF2_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A32	BUFFER	SIMD 8	1	MSD1R_DWAF2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	A32	BUFFER	SIMD 8	1	MSD1W_DWAF2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	A32	BUFFER	SIMD 16	1	MSD1R_DWAF2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16
W	A32	BUFFER	SIMD 16	1	MSD1W_DWAF2	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	A32	BUFFER	SIMD 4x2	1	MSD1R_DWAF3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	A32	BUFFER	SIMD 4x2	1	MSD1W_DWAF3_4x2	{Opt} MH1_A32	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	A32	BUFFER	SIMD 8	1	MSD1R_DWAF3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	A32	BUFFER	SIMD 8	1	MSD1W_DWAF3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	A32	BUFFER	SIMD 16	1	MSD1R_DWAF3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	A32	BUFFER	SIMD 16	1	MSD1W_DWAF3	{Opt} MH1_A32_PSM	MAP32b_USU_SIMD16	MDP_AOP16_DW2	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_DWAF2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_DWAF2_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	A64	BUFFER	SIMD 8	1	MSD1R_A64_DWAF2	{Forbidden}	MAP64b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	A64	BUFFER	SIMD 8	1	MSD1W_A64_DWAF2	{Forbidden}	MAP64b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	A64	BUFFER	SIMD 4x2	1	MSD1R_A64_DWAF3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	A64	BUFFER	SIMD 4x2	1	MSD1W_A64_DWAF3_4x2	{Forbidden}	MAP64b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	A64	BUFFER	SIMD8	1	MSD1R_A64_DWAF3	{Forbidden}	MAP64b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	A64	BUFFER	SIMD8	1	MSD1W_A64_DWAF3	{Forbidden}	MAP64b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}

DWord Scaled Untyped Atomic Float Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	STRBUF,NULL	4x2	1	(Base+(U{Slot}*Pitch)+V{Slot})→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	STRBUF,NULL	8, 16	1	(Base+(U{Slot}*Pitch)+V{Slot})→DW[0]	Surface	SMPSM

This message performs atomic float operations on untyped surfaces using 8 or 16 offsets. One DWord is accessed beginning at the byte address determined. Each U offset is scaled by Pitch. Both Surface Pitch and V offset must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	STRBUF,NULL	SIMD4x2	1	MSD1R_DWAF2_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	STRBUF,NULL	SIMD4x2	1	MSD1W_DWAF2_4x2	{Opt}MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	STRBUF,NULL	SIMD8	1	MSD1R_DWAF2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD8	1	MSD1W_DWAF2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	STRBUF,NULL	SIMD16	1	MSD1R_DWAF2	{Opt}MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_DW_SIMD16	MDP_DW_SIMD16

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	STRBUF,NULL	SIMD16	1	MSD1W_DWAF2	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_DW_SIMD16	{Forbidden}
R	BTS	STRBUF,NULL	SIMD4x2	1	MSD1R_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	BTS	STRBUF,NULL	SIMD4x2	1	MSD1W_DWAF3_4x2	{Opt} MH_IGNORE	MAP32b_USUV_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD8	1	MSD1R_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	STRBUF,NULL	SIMD8	1	MSD1W_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	BTS	STRBUF,NULL	SIMD16	1	MSD1R_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_DW2	MDP_DW_SIMD16
W	BTS	STRBUF,NULL	SIMD16	1	MSD1W_DWAF3	{Opt} MH1_BTS_PSM	MAP32b_USUV_SIMD16	MDP_AOP16_DW2	{Forbidden}

DWord Typed Atomic Integer Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	1D, 2D, 3D, CUBE	4x2	1	(Surface[U, V, R, LOD])→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	BUFFER, NULL	4x2	1	(Surface[U])→DW[0]	Surface	SM4x2
DW	DW	R/W	BTS	1D, 2D, 3D, CUBE	8	1	(Surface[U, V, R, LOD])→DW[0]	Surface	SGPSM
DW	DW	R/W	BTS	BUFFER, NULL	8	1	(Surface[U])→DW[0]	Surface	SGPSM

This message performs atomic integer operations on typed surfaces using 8 offsets. The surface format must be R32_UINT or R32_SINT, and the surface type must be SURFTYPE_1D, SURFTYPE_2D, SURFTYPE_3D, SURFTYPE_CUBE, or SURFTYPE_BUFFER. One DWord is accessed beginning at the byte address determined by the address payload and the surface format. Each access is a DWord and must be DWord-aligned.

The execution mask bits may disable accesses on the corresponding SIMD slots. Out-of-bounds accesses are dropped or return zero. The semantics depend on the address model, as listed in the above table.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	1D, 2D, 3D, CUBE	SIMD 4x2	1	MSD1R_DWTAI1_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	BTS	1D, 2D, 3D, CUBE	SIMD 4x2	1	MSD1W_DWTAI1_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	{Forbidden}	{Forbidden}
R	BTS	BUFFER, NULL	SIMD 4x2	1	MSD1R_DWTAI1_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	MDP_AOP4x2_DW1
W	BTS	BUFFER, NULL	SIMD 4x2	1	MSD1W_DWTAI1_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	{Forbidden}	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE	SIMD 8	1	MSD1R_DWTAI1	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	BTS	1D, 2D, 3D, CUBE	SIMD 8	1	MSD1W_DWTAI1	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	{Forbidden}	{Forbidden}
R	BTS	BUFFER, NULL	SIMD 8	1	MSD1R_DWTAI1	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	{Forbidden}	MDP_DW_SIMD8
W	BTS	BUFFER, NULL	SIMD 8	1	MSD1W_DWTAI1	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	{Forbidden}	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE	SIMD 4x2	1	MSD1R_DWTAI2_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	1D, 2D, 3D, CUBE	SIMD 4x2	1	MSD1W_DWTAI2_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	BUFFER, NULL	SIMD 4x2	1	MSD1R_DWTAI2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	BUFFER, NULL	SIMD 4x2	1	MSD1W_DWTAI2_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	1D, 2D, 3D,	SIMD 8	1	MSD1R_DWTAI2	{Opt} MH1_BTS_	MAP32b_TS_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
		CUBE				PSM			
W	BTS	1D, 2D, 3D, CUBE	SIMD8	1	MSD1W_DWTAI2	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	BUFFER, NULL	SIMD8	1	MSD1R_DWTAI2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	BUFFER, NULL	SIMD8	1	MSD1W_DWTAI2	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_DW_SIMD8	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE	SIMD4x2	1	MSD1R_DWTAI3_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	BTS	1D, 2D, 3D, CUBE	SIMD4x2	1	MSD1W_DWTAI3_4x2	{Opt} MH_IGNORE	MAP32b_TS_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	BUFFER, NULL	SIMD4x2	1	MSD1R_DWTAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	MDP_AOP4x2_DW1
W	BTS	BUFFER, NULL	SIMD4x2	1	MSD1W_DWTAI3_4x2	{Opt} MH_IGNORE	MAP32b_USU_SIMD4x2	MDP_AOP4x2_DW2	{Forbidden}
R	BTS	1D, 2D, 3D, CUBE	SIMD8	1	MSD1R_DWTAI3	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	1D, 2D, 3D, CUBE	SIMD8	1	MSD1W_DWTAI3	{Opt} MH1_BTS_PSM	MAP32b_TS_SIMD8	MDP_AOP8_DW2	{Forbidden}
R	BTS	BUFFER, NULL	SIMD8	1	MSD1R_DWTAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	MDP_DW_SIMD8
W	BTS	BUFFER, NULL	SIMD8	1	MSD1W_DWTAI3	{Opt} MH1_BTS_PSM	MAP32b_USU_SIMD8	MDP_AOP8_DW2	{Forbidden}

Programming Note	
Context:	DWord Typed Atomic Integer Messages
The U address payload specifies a pixel index into the surface, which is multiplied by the **Surface Pitch** from the Surface State to generate the byte address. That final byte address must be Dword aligned.	

DWord Atomic Counter Messages

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R/W	BTS	Any	4x2	1	(Surface Append Buffer)→DW[0]	Ignored	SM4x2
DW	DW	R/W	BTS	Any	8	1	(Surface Append Buffer)→DW[0]	Ignored	SGPSM, SG

This message performs atomic integer operations on the "Append Counter" associated with the surface, using 8 offsets. One DWord is accessed beginning at the byte address of the surface's auxiliary buffer address, which must be DWord-aligned.

For Append Counter Operations there is no address payload: the address is provided by the append counter field in the surface state. The data payloads are the same as untyped atomic integer operations.

When accessing a surface with this message, if the Auxiliary Surface Mode of the surface state is not AUX_APPEND, the access is treated as out of bounds with the writes being ignored and the reads returning 0.

The execution mask bits may disable accesses on the corresponding SIMD slots.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	Any	SIMD 4x2	1	MSD1R_DWAC1_4x2	MH_IGNORE	{Forbidden}	{Forbidden}	MDP_AOP4x2_DW1
W	BTS	Any	SIMD 4x2	1	MSD1W_DWAC1_4x2	MH_IGNORE	{Forbidden}	{Forbidden}	{Forbidden}
R	BTS	Any	SIMD 8	1	MSD1R_DWAC1	MH1_BTS_PSM	{Forbidden}	{Forbidden}	MDP_DW_SIMD8
W	BTS	Any	SIMD 8	1	MSD1W_DWAC1	MH1_BTS_PSM	{Forbidden}	{Forbidden}	{Forbidden}
R	BTS	Any	SIMD 4x2	1	MSD1R_DWAC2_4x2	{Forbidden}	{Forbidden}	MDP_AOP4x2_DW1	MDP_AOP4x2_DW1
W	BTS	Any	SIMD 4x2	1	MSD1W_DWAC2_4x2	{Forbidden}	{Forbidden}	MDP_AOP4x2_DW1	{Forbidden}
R	BTS	Any	SIMD 8	1	MSD1R_DWAC2	{Forbidden}	{Forbidden}	MDP_DW_SIMD8	MDP_DW_SIMD8
W	BTS	Any	SIMD 8	1	MSD1W_DWAC2	{Forbidden}	{Forbidden}	MDP_DW_SIMD8	{Forbidden}

Programming Note	
Context:	DWord Atomic Counter Messages

- The unary atomic integer operations require a message header (so that the message data on the OBUS is not empty).
- The current implementation of the binary atomic integer operations forbids a message header, so the Pixel Sample Mask is defaulted to be fully enabled.
- The Append Counter cannot be used with an MSAA auxiliary surface.

Media Surface Read/Write Messages

The media block and transpose messages allow direct read/write accesses to media surfaces. These messages operate on rectangular blocks of pixel data.

Media surfaces are 2D types with some media-specific characteristics (for example, vertical offset handling for interlaced frames). The surface state specifies all the parameters. There are some limitations on what data surface formats are supported by data ports, when compared to the Sampler, Render Cache, and the fixed function Video encode and decode engines. See the specific messages and Addressing 2D Media Surfaces in this volume for more details.

Transpose Read Message

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R	BTS	2D, NULL	Height	Width	(Surface[X, Y])→DW[0]	Media	Ignored

This message enables a rectangular block of DWords to be read from the source surface and written in transposed order into the GRF. The message specifies as inputs a signed (X, Y) coordinate into the 2D surface and a block size (Width, Height).

The execution mask is ignored. Out-of-bounds reads return a replicated boundary pixel. More details on bounds checking is described in Addressing 2D Media Surfaces in this volume.

Applications:

Transpose a 2D media block of 32bpe pixels.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	2D, NULL	1, 2, 4, 8 Rows	{1, 2, 4, 8} Width	MSD1R_TT	MH1_T	{Forbidden}	{Forbidden}	{1, 2, 4, 8} MDP_DW_SIMD8

The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Accesses are allowed to SURFTYPE_NULL and will return 0. More details on 2D surfaces used by this message is described in Addressing 2D Media Surfaces in this volume

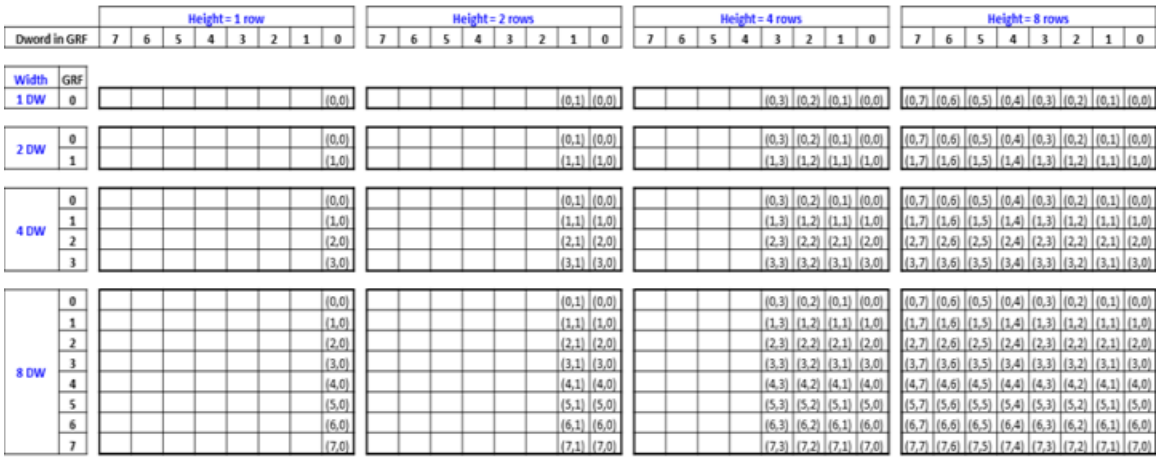
Vertical stride, offset, and boundary clamping media surface state parameters are supported. The surface format must be 32bpe (Dword size) as described in 2D Media Surface Formats. The raw data from the surface is returned to the thread without any format conversion operation.

Block Width and Block Height

The Block Width (in Dwords) and Block Height (in rows) are specified in the message header.

The layout of the data payload depends on the Block Height and Block Width. Each row's first Dword is aligned to the corresponding Dword of the first register, and every next Dword in the X dimension of the block is aligned to the corresponding Dword of the next register. The figure below illustrates the layout of the data returned in the register in (x,y) offsets from the (X,Y) start of the block. Dwords not used in a GRF are not modified.

Layout of Transposed Rows in Registers



Programming Note	
Context:	Transpose Read Message

- Tile X is not supported: the supported tiling modes are Linear and Tile Y/YF/YS.
- The surface base address must be 32-byte aligned.
- The surface width must be a multiple of DWords.
- Pitch must be a multiple of 64 bytes when the surface is linear.
- The **X Offset** must be a multiple of the block width in bytes. The **Y Offset** must be a multiple of the block height.
- Transpose Read is not supported on compressible media surfaces.

Media Block Read/Write Messages

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R / W	BTS	2D, NULL	1 block	Width x Height	(Surface[X, Y])→DW[0]	Media	Ignored

The read form of this message enables a rectangular block of data samples to be read from the source surface and written into the GRF. The write form enables data from the GRF to be written to a rectangular block. The message specifies as inputs a signed (X, Y) coordinate into the 2D surface and a block size (Width, Height).

The execution mask is ignored. Out-of-bounds writes are dropped. Out-of-bounds reads return a replicated boundary pixel. More details on bounds checking is described in Addressing 2D Media Surfaces in this volume.

Applications:

Block reads/writes for media

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	2D, NULL	1 block	Height rows x Width bytes	MSD1R_MB	MH_MB	{Forbidden}	{Forbidden}	{1-8} MDP_HW_B
W	BTS	2D, NULL	1 block	Height rows x Width bytes	MSD1W_MB	MH_MB	{Forbidden}	{1-8} MDP_HW_B	{Forbidden}
R	BTS	2D, NULL	1 block	Height rows x Width bytes	MSD_SC_MB	MH_MB	{Forbidden}	{Forbidden}	{1-8} MDP_HW_B

The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Accesses are allowed to SURFTYPE_NULL, reads will return 0 and writes will be ignored. Only non-IA-coherent surfaces may be used with compressed media surfaces. More details on 2D surfaces used by this message is described in Addressing 2D Media Surfaces in this volume.

See 2D Media Surface Formats in this volume for limitations on surface formats supported by this message.

Block Width and Block Height

The maximum Block Height is limited by the Block Width. The maximum data size supported in this message is 64 Dwords (256 Bytes). Blocks wider than 32 Bytes are supported only with write operations with either linear and Tile X surfaces.

Block Width (bytes)	Block Height (rows)	Tile Modes Supported
1-4	1-64	Linear, TileX, TileY/YF/YS
5-8	1-32	Linear, TileX, TileY/YF/YS
9-16	1-16	Linear, TileX, TileY/YF/YS
17-32	1-8	Linear, TileX, TileY/YF/YS
33-64	1-4	Linear, TileX

The layout of the read and write data payload depends on the Block Height and Block Width. The data is aligned to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the Block Width. The figure below illustrates the layout of the rows in the registers.

Media Block Row Layout in Registers

		Width = 1-4								Width = 5-8				Width = 9-16		Width = 17-32	Width = 33-64
	Dword	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
GRF	0	8	7	6	5	4	3	2	1	4	3	2	1	2	1	1	1
	1	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	
	2	24	23	22	21	20	19	18	17	12	11	10	9	6	5	3	2
	3	32	31	30	29	28	27	26	25	16	15	14	13	8	7	4	
	4	40	39	38	37	36	35	34	33	20	19	18	17	10	9	5	3
	5	48	47	46	45	44	43	42	41	24	23	22	21	12	11	6	
	6	56	55	54	53	52	51	50	49	28	27	26	25	14	13	7	4
	7	64	63	62	61	60	59	58	57	32	31	30	29	16	15	8	

Programming Note	
Context:	Media Block Read/Write Messages

Restrictions Media Block Read and Write:

- The surface base address must be 32-byte aligned.
- Pitch must be a multiple of 64 bytes when the surface is linear.
- For YUV422 formats, the block width and offset must be pixel pair aligned (i.e. DWord-aligned).
- The block width and offset must be aligned to the size of pixels stored in the surface. For a surface with 8bpe pixels for example, the block width and offset can be byte-aligned. For a surface with 16bpe pixels, it is word-aligned.

Restrictions Media Block Read:

- For media block reads, the Block Width must be 32 or less.

Programming Note	
Context:	Media Block Read/Write Messages

Restrictions Media Block Write:

- For media block writes, both X Offset and Block Width must be DWord-aligned.
- Media block writes to Linear or TileX surfaces must have a height of 16 or less.
- Media block writes are not supported on compressible media surfaces.

Pixel Masked Media Block Write Message

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	W	BTS	2D, NULL	1 block	Width x Height	(Surface[X, Y])→DW[0]	Media	Ignored

This message conditionally writes a rectangular block of pixels from GRF data. The message specifies as inputs a signed (X, Y) coordinate into the 2D surface with a fixed block size of 4 rows by 8 Dwords, and a Pixel Mask to enable writing of the individual Dwords.

The execution mask is ignored. Out-of-bounds writes are dropped. Media bounds checking is more fully described in Addressing 2D Media Surfaces in this volume.

Applications:

Block reads/writes for media

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	2D, NULL	1 block	4 rows x 8 DW	MSD1W_MB	MH_MBPM	{Forbidden}	{4} MDP_HW_B	{Forbidden}

The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Accesses are allowed to SURFTYPE_NULL, writes will be ignored. More details on 2D surfaces used by this message is described in Addressing 2D Media Surfaces in this volume

See 2D Media Surface Formats in this volume for limitations on surface formats supported by this message.

All tile modes are supported: Linear, TileX, TileY/YF/YS.

The data payload is aligned with the upper left pixel going into the least significant bits of the first register, with 1 row per register. The Pixel Mask applies to the 2x2 square tiles (UL, UR, LL, LR), which themselves tiled (UL, UR, LL, LR) and then repeated on the right for the remaining 16-bits to cover a 4 row 8 column area. When the Pixel Mask bit is set, the Dword is written. The figure below illustrates the layout of the data and corresponding mask bits.

Data and Pixel Mask Layout

Dword	7	6	5	4	3	2	1	0

GRF Row Data									
	0	7	6	5	4	3	2	1	0
	1	15	14	13	12	11	10	9	8
	2	23	22	21	20	19	18	17	16
	3	31	30	29	28	27	26	25	24

Corresponding Pixel Mask Bits								
21	20	17	16	5	4	1	0
23	22	19	18	7	6	3	2
29	28	25	24	13	12	9	8
31	30	27	26	15	14	11	10

Programming Note	
Context:	Pixel Masked Media Block Write Message

- X Offset must be DWord-aligned.
- Pitch must be a multiple of 64 bytes when the surface is linear.
- The surface base address must be 32-byte aligned.
- Media block writes are not supported on compressible media surfaces.

Byte Masked Media Block Write Message

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	W	BTS	2D, NULL	1 block	Width x Height	(Surface[X, Y])→DW[0]	Media	Ignored

This message conditionally writes a rectangular block of pixels from GRF data. The message specifies as inputs a signed (X, Y) coordinate into the 2D surface with block size (Width, Height), and a Byte Mask to enable writing of the individual bytes.

The execution mask is ignored. Out-of-bounds writes are dropped. Media bounds checking is more fully described in Addressing 2D Media Surfaces in this volume.

Applications:

Block writes for media

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	2D, NULL	1 block	Height rows x Width bytes	MSD1W_MB	MH_MBBM	{Forbidden}	{1-8} MDP_HW_B	{Forbidden}

The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Accesses are allowed to SURFTYPE_NULL, writes will be ignored. More details on 2D surfaces used by this message is described in Addressing 2D Media Surfaces in this volume.

See 2D Media Surface Formats in this volume for limitations on surface formats supported by this message.

Block Width and Block Height

The maximum Block Height is limited by the Block Width. The maximum data size supported in this message is 64 Dwords (256 Bytes).

Block Width (bytes)	Block Height (rows)	Tile Modes Supported
1-4	1-64	Linear, TileX, TileY/YF/YS
5-8	1-32	Linear, TileX, TileY/YF/YS
9-16	1-16	Linear, TileX, TileY/YF/YS
17-32	1-8	Linear, TileX, TileY/YF/YS

The data payload is aligned with the upper left pixel going into the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the Block Width. The Byte Mask applies horizontally to each row of output. When the Byte Mask bit is set, the byte is written. The figure below illustrates the layout of the row data and corresponding mask bits.

Row and Byte Mask Layout

Dword		Width = 1-4								Width = 5-8								Width = 9-16								Width = 17-32							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
GRF	0	8	7	6	5	4	3	2	1	4	3	2	1					2	1							1							
	1	16	15	14	13	12	11	10	9	8	7	6	5					4	3							2							
	2	24	23	22	21	20	19	18	17	12	11	10	9					6	5							3							
	3	32	31	30	29	28	27	26	25	16	15	14	13					8	7							4							
	4	40	39	38	37	36	35	34	33	20	19	18	17					10	9							5							
	5	48	47	46	45	44	43	42	41	24	23	22	21					12	11							6							
	6	56	55	54	53	52	51	50	49	28	27	26	25					14	13							7							
	7	64	63	62	61	60	59	58	57	32	31	30	29					16	15							8							
Byte Mask		Bits 3:0 for each Row								Bits 7:0 for each Row								Bits 15:0 for each Row								Bits 31:0 for each Row							

Programming Note	
Context:	Byte Masked Media Block Write Message

- Linear or TileX surfaces must have a block height <= 16.
- Both X Offset and Block Width must be DWord-aligned.
- The surface base address must be 32-byte aligned.
- Pitch must be a multiple of 64 bytes when the surface is linear.
- Media block writes are not supported on compressible media surfaces.

Merged Media Block Read/Write Messages

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R	BTS	2D, NULL	1 block	Width x Height	(Surface[X, Y])→DW[0]	Media	Ignored

This message enables a rectangular block of data samples to be read from the source surface, and written into the GRF in a spread-out form without modifying any other portions of the GRF. The message specifies as inputs a signed (X, Y) coordinate into the 2D surface, a block size (Width, Height), a Register Pitch Control, and a Sub-Register Offset.

The execution mask is ignored. Out-of-bounds reads return a replicated boundary pixel. Media bounds checking is described in Addressing 2D Media Surfaces in this volume.

Applications:

Allow software to assemble multiple media block reads directly into a shared GRF register set. Multiple media block read messages sharing the same Register Pitch Control but with different Sub-Register Offset can fill in the same set of GRF registers with interleaved rows of media blocks.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	2D, NULL	1 block	Height rows x Width bytes	MSD1R_MB	MH_MBM	{Forbidden}	{Forbidden}	{1-31} MDP_HW_B

The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Reads are allowed to SURFTYPE_NULL and will return 0. More details on 2D surfaces used by this message are described in Addressing 2D Media Surfaces in this volume.

See 2D Media Surface Formats in this volume for limitations on surface formats supported by this message.

Block Width and Block Height

The layout of the read data payload depends on the Block Height, Block Width, Register Pitch, and Sub-Register Offset. The upper left pixel is aligned to the least significant bits of the first register, and the register pitch is equal to the next multiple of the Register Pitch Control and Data Width. Finally, the data is aligned to the Sub-Register Offset of the first register.

The maximum Block Height is limited by the Block Width. The maximum packed data size supported in this message is 64 Dwords (256 Bytes).When Register Pitch Control is 4 rows and the Block Width <= 8, the maximum Block Height is additionally limited by the maximum allowed message response of 31 GRF registers.

Block Width (bytes)	Register Pitch Control	Sub-Register Offset	Block Height (rows) Supported	Tile Modes Supported
1-4	0	0,4,8,12,16,20,24,28	1	Linear, TileX, TileY/YF/YS
	2	0,4	1-64	Linear, TileX, TileY/YF/YS
	2	16,20	1-2	Linear, TileX, TileY/YF/YS
	4	0,4,8,12	1-62	Linear, TileX, TileY/YF/YS
5-8	0	0,8,16,24	1	Linear, TileX, TileY/YF/YS
	2	0,8	1-32	Linear, TileX, TileY/YF/YS

Block Width (bytes)	Register Pitch Control	Sub-Register Offset	Block Height (rows) Supported	Tile Modes Supported
	4	0,8	1-31	Linear, TileX, TileY/YF/YS
9-16	0	0,16	1	Linear, TileX, TileY/YF/YS
	2	16	1-16	Linear, TileX, TileY/YF/YS
	4	16	1-16	Linear, TileX, TileY/YF/YS
17-32	2	0	1-8	Linear, TileX, TileY/YF/YS
	4	0	1-8	Linear, TileX, TileY/YF/YS

Certain combinations of Register Pitch and Sub-Register Offset allow interleaving multiple rows into one block. When only 1 GRF is being updated, then additional Sub-Register Offset combinations are supported. The figures below illustrate for each of the block width cases, the layout of the rows in the registers and which combinations of Register Pitch Control and Sub-Register Offset are supported.

Two-way Row Interleave into Registers

Register Pitch Control	2	Sub-Register Offset (SRO) selections			
		SRO = 0	SRO = 0	SRO = 0	SRO = 0
		SRO = 4	SRO = 8	SRO = 16	GRF+1, SRO = 0

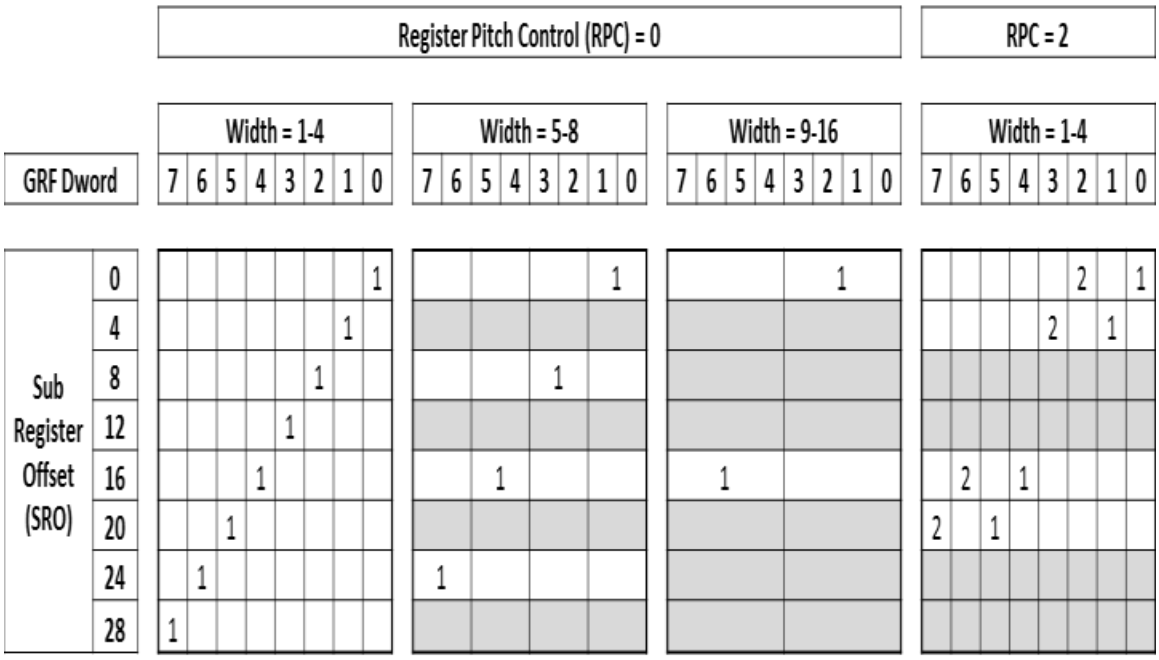
		Width = 1-4								Width = 5-8								Width = 9-16								Width = 17-32							
Dword		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
GRF	0	4	4	3	3	2	2	1	1	2	2	1	1					1	1							1							
	1	8	8	7	7	6	6	5	5	4	4	3	3					2	2							1							
	2	12	12	11	11	10	10	9	9	6	6	5	5					3	3							2							
	3	16	16	15	15	14	14	13	13	8	8	7	7					4	4							2							
	4	20	20	19	19	18	18	17	17	10	10	9	9					5	5							3							
	5	24	24	23	23	22	22	21	21	12	12	11	11					6	6							3							
	6	28	28	27	27	26	26	25	25	14	14	13	13					7	7							4							
	7	32	32	31	31	30	30	29	29	16	16	15	15					8	8							4							
	8	36	36	35	35	34	34	33	33	18	18	17	17					9	9							5							
	9	40	40	39	39	38	38	37	37	20	20	19	19					10	10							5							
	10	44	44	43	43	42	42	41	41	22	22	21	21					11	11							6							
	11	48	48	47	47	46	46	45	45	24	24	23	23					12	12							6							
	12	52	52	51	51	50	50	49	49	26	26	25	25					13	13							7							
	13	56	56	55	55	54	54	53	53	28	28	27	27					14	14							7							
	14	60	60	59	59	58	58	57	57	30	30	29	29					15	15							8							
	15	64	64	63	63	62	62	61	61	32	32	31	31					16	16							8							

Four-way Row Interleave into Registers

Register Pitch Control	4	Sub-Register Offset (SRO) selections			
		SRO = 0	SRO = 0	SRO = 0	SRO = 0
		SRO = 4	SRO = 8	SRO = 16	GRF+1, SRO = 0
		SRO = 8	SRO = 16	GRF+1, SRO = 0	GRF+2, SRO = 0
		SRO = 12	SRO = 24	GRF+1, SRO = 16	GRF+3, SRO = 0

Dword		Width = 1-4								Width = 5-8								Width = 9-16								Width = 17-32							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
GRF	0	2	2	2	2	1	1	1	1	1	1	1	1					1	1							1							
	1	4	4	4	4	3	3	3	3	2	2	2	2					1	1							1							
	2	6	6	6	6	5	5	5	5	3	3	3	3					2	2							1							
	3	8	8	8	8	7	7	7	7	4	4	4	4					2	2							1							
	4	10	10	10	10	9	9	9	9	5	5	5	5					3	3							2							
	5	12	12	12	12	11	11	11	11	6	6	6	6					3	3							2							
	6	14	14	14	14	13	13	13	13	7	7	7	7					4	4							2							
	7	16	16	16	16	15	15	15	15	8	8	8	8					4	4							2							
	8	18	18	18	18	17	17	17	17	9	9	9	9					5	5							3							
	9	20	20	20	20	19	19	19	19	10	10	10	10					5	5							3							
	10	22	22	22	22	21	21	21	21	11	11	11	11					6	6							3							
	11	24	24	24	24	23	23	23	23	12	12	12	12					6	6							3							
	12	26	26	26	26	25	25	25	25	13	13	13	13					7	7							4							
	13	28	28	28	28	27	27	27	27	14	14	14	14					7	7							4							
	14	30	30	30	30	29	29	29	29	15	15	15	15					8	8							4							
	15	32	32	32	32	31	31	31	31	16	16	16	16					8	8							4							
	16	34	34	34	34	33	33	33	33	17	17	17	17					9	9							5							
	17	36	36	36	36	35	35	35	35	18	18	18	18					9	9							5							
	18	38	38	38	38	37	37	37	37	19	19	19	19					10	10							5							
	19	40	40	40	40	39	39	39	39	20	20	20	20					10	10							5							
	20	42	42	42	42	41	41	41	41	21	21	21	21					11	11							6							
	21	44	44	44	44	43	43	43	43	22	22	22	22					11	11							6							
	22	46	46	46	46	45	45	45	45	23	23	23	23					12	12							6							
	23	48	48	48	48	47	47	47	47	24	24	24	24					12	12							6							
	24	50	50	50	50	49	49	49	49	25	25	25	25					13	13							7							
	25	52	52	52	52	51	51	51	51	26	26	26	26					13	13							7							
	26	54	54	54	54	53	53	53	53	28	28	28	28					14	14							7							
	28	56	56	56	56	55	55	55	55	28	28	28	28					14	14							7							
	28	58	58	58	58	57	57	57	57	29	29	29	29					15	15							8							
	29	60	60	60	60	59	59	59	59	30	30	30	30					15	15							8							
	30	62	62	62	62	61	61	61	61	31	31	31	31					16	16							8							
	31																	16	16							8							

Merging Rows into 1 Register



Programming Note	
Context:	Merged Media Block Read/Write Messages

- The surface base address must be 32-byte aligned.
- Pitch must be a multiple of 64 bytes when the surface is linear.
- For YUV422 formats, the block width and offset must be pixel pair aligned (i.e. DWord-aligned).
- The block width and offset should be aligned to the size of pixels stored in the surface. For a surface with 8bpp pixels for example, the block width and offset can be byte-aligned. For a surface with 16bpp pixels, it is word-aligned.

Other Data Port Messages

This section describes the remaining data port messages that do not fit into one of the other groupings.

Read Surface Info Message

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
B	B	R	BTS	Any	1	1		Ignored	Ignored

This message is used to determine information about a surface MIP level. It returns the surface format, type, width, depth, height of the specified surface address.

The LOD is in-bounds if address payload's LOD < surface's MIPCount and if the surface's MinLOD + address payload's LOD < 15. If the LOD is not in-bounds then 0 is returned for the width, height, and depth values.

The execution mask is ignored. No bounds checking is performed on the U, V, or R address components in the address payload.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	Any	1	1	MSD_RSI	{Forbidden}	MAP32b_RSI	{Forbidden}	MDP_RSI

There is no message header for this message type.

The 64-bit Instruction Base Address returned in the Writeback Payload is specified as a 48-bit state base address and is extended to 64 bits in HW, so SW can read it for conversion of 64-bit instruction pointers.

Memory Fence Message

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Ignored	Ignored

A memory fence message issued by a thread causes further messages issued by the thread to be blocked until all previous data port messages have completed, or the results can be globally observed from the point of view of other threads in the system. This includes both read and write messages.

The execution mask is ignored. No bounds checking is performed.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
N/A	N/A	N/A	N/A	N/A	MSD_MEMFENCE	MH_IGNORE	{Forbidden}	{Forbidden}	MDP_DW_SIMD8

The memory fence message signals completion by returning data into the writeback register. The data returned in the writeback register is undefined. When an instruction reads the writeback register value, then this thread is blocked until all previous data port messages are globally observable.

A memory fence memory is typically performed prior to a Gateway Barrier message (which synchronizes threads in a group) so that when the barrier releases the threads, each thread is ensured its next data port message is ordered after the memory fence. See Read/Write Ordering in this volume for more details.

Global observability of SLM memory means the SLM memory is visible to all the threads in the thread group that have access to that memory. Global observability of the coherent L3 cache means that those portions of the L3 cache are observable both to graphics threads and to the rest of the system. Global observability of non-coherent L3 cache means those portions of the L3 cache are observable to graphics threads but not necessarily to the rest of the system.

The memory fence message descriptor also has controls to flush (invalidate) various Gen caches. These controls are generally not used.

Programming Note	
Context:	Memory Fence Message
• The L3 has a few different partitioning schemes. In some cases RW data, Constant data and/or texture data can be mixed in the same partition. If a flush is needed for any data type in the partition the entire partition is flushed. • When a cache flush control is specified to the L3 cache, the completion of the memory fence message does not mean that the cache flush has completed. When a thread needs the flush to be completed before continuing execution, the thread should issue a data port read or atomic operation (to any global address) after the cache flush. The cache flush will be completed when this global memory read or atomic operation completes.	

Workaround-Memory Fence Message Commit Enable

Message-Specific Descriptors

All the operations supported on the 4 data ports (Data Port 0, 1, 2, and RO), and over the 4 address models (BTS, SLM, A32, A64) are described by the Message Descriptors.

All the message descriptors indicate whether a Message Header is present, and the Message Type. The decoding of each message descriptor is specific to the message type and the data port it is sent to.

Data Port 0, 1, and RO usually provide a Binding Table Index (BTI) to specify a surface in the BTS address model. The special entry 255 is used to reference Stateless A32 or A64 address model, and the special entry 254 is used to reference the SLM address model.

The special entry 252 is used to reference bindless resource operation. When 252 is used, the surface is identified by the SSO (Surface State Offset) field. That field is encoded in the SENDS instruction's extended descriptor using indirect register addressing.

Data Port Bindless Surface Extended Message Descriptor

Data Port 2 Message Descriptors
Data Port 2 messages do not have a BTI field, and encode whether the access is to SLM or A32 memory based on the settings of the Message Header Present and SBSO (Sideband Scaled Offset) fields. The SBSO field is encoded in the SENDS instruction's extended descriptor on bits [31:16].

Data Port 2 Extended Message Descriptor

Data Port 0 Message Specific Descriptors

This section contains the message specific descriptors for Data Port 0.

MT_DP0 - Data Port 0 Message Types

Scattered Read/Write Messages

Message Descriptor
MSD0R_BS - Byte Scattered Read MSD
MSD0W_BS - Byte Scattered Write MSD
MSD0R_DWS - Dword Scattered Read MSD
MSD0W_DWS - Dword Scattered Write MSD

Block Read/Write Messages

Message Descriptor
MSD0R_OWB - Oword Block Read MSD
MSD0W_OWB - Oword Block Write MSD
MSD0R_OWAB - Oword Aligned Block Read MSD
MSD0R_OWDB - Oword Dual Block Read MSD
MSD0W_OWDB - Oword Dual Block Write MSD
MSD0R_HWB - Scratch Block Read MSD
MSD0W_HWB - Scratch Block Write MSD

Other Data Port Messages

Message Descriptor
MSD_MEMFENCE - Memory Fence MSD

Data Port 1 Message Specific Descriptors

This section contains the message specific descriptors for Data Port 1.

MT_DP1 - Data Port 1 Message Types

A64 Scattered Read/Write

Descriptor
MSD1R_A64_BS - A64 Byte Scattered Read MSD
MSD1W_A64_BS - A64 Byte Scattered Write MSD
MSD1R_A64_DWS - A64 Dword Scattered Read MSD
MSD1W_A64_DWS - A64 Dword Scattered Write MSD
MSD1R_A64_QWS - A64 Qword Scattered Read MSD
MSD1W_A64_QWS - A64 Qword Scattered Write MSD

A64 Block Read/Write

Descriptor
MSD1R_A64_OWB - A64 Oword Block Read MSD
MSD1W_A64_OWB - A64 Oword Block Write MSD
MSD1R_A64_OWUB - A64 Oword Unaligned Block Read MSD
MSD1R_A64_OWDB - A64 Oword Dual Block Read MSD
MSD1W_A64_OWDB - A64 Oword Dual Block Write MSD
MSD1R_A64_HWB - A64 Hword Block Read MSD
MSD1W_A64_HWB - A64 Hword Block Write MSD

Surface Read/Write

Descriptor
MSD1R_US - Untyped Surface Read MSD
MSD1W_US - Untyped Surface Write MSD
MSD1R_A64_US - A64 Untyped Surface Read MSD
MSD1W_A64_US - A64 Untyped Surface Write MSD
MSD1R_TS - Typed Surface Read MSD
MSD1W_TS - Typed Surface Write MSD

Media Messages

Descriptor
MSD1R_TT - Media Transpose Read MSD
MSD1R_MB - Media Block Read MSD
MSD1W_MB - Media Block Write MSD

A32 Dword Untyped Atomic Operations

Descriptor
MSD1R_DWAI1 - Dword Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_DWAI2 - Dword Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_DWAI3 - Dword Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_DWAI1 - Dword Untyped Atomic Integer Unary Write Only Operation MSD
MSD1W_DWAI2 - Dword Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_DWAI3 - Dword Untyped Atomic Integer Trinary Write Only Operation MSD
MSD1R_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Descriptor
MSD1W_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD
MSD1R_DWAF2 - Dword Untyped Atomic Float Binary with Return Data Operation MSD
MSD1R_DWAF3 - Dword Untyped Atomic Float Trinary with Return Data Operation MSD
MSD1W_DWAF2 - Dword Untyped Atomic Float Binary Write Only Operation MSD
MSD1W_DWAF3 - Dword Untyped Atomic Float Trinary Write Only Operation MSD
MSD1R_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD
MSD1R_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD
MSD1W_DWAF2_4x2 - Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD
MSD1W_DWAF3_4x2 - Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

A64 Dword Untyped Atomic Operations

Descriptor
MSD1R_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD
MSD1W_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD
MSD1R_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD
MSD1W_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD
MSD1R_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary with Return Data Operation MSD
MSD1R_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary with Return Data Operation MSD
MSD1W_A64_DWAF2 - A64 Dword Untyped Atomic Float Binary Write Only Operation MSD
MSD1W_A64_DWAF3 - A64 Dword Untyped Atomic Float Trinary Write Only Operation MSD
MSD1R_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary with Return Data Operation MSD
MSD1R_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary with Return Data Operation MSD
MSD1W_A64_DWAF2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Binary Write Only Operation MSD
MSD1W_A64_DWAF3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Float Trinary Write Only Operation MSD

A64 Qword Untyped Atomic Operations

Descriptor
MSD1R_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD
MSD1W_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD
MSD1R_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD
MSD1R_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD
MSD1R_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD
MSD1W_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD
MSD1W_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD
MSD1W_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Dword Typed Atomic Operations

Descriptor
MSD1R_DWTAI1 - Dword Typed Atomic Integer Unary with Return Data Operation MSD
MSD1R_DWTAI2 - Dword Typed Atomic Integer Binary with Return Data Operation MSD
MSD1R_DWTAI3 - Dword Typed Atomic Integer Trinary with Return Data Operation MSD
MSD1W_DWTAI1 - Dword Typed Atomic Integer Unary Write Only Operation MSD
MSD1W_DWTAI2 - Dword Typed Atomic Integer Binary Write Only Operation MSD
MSD1W_DWTAI3 - Dword Typed Atomic Integer Trinary Write Only Operation MSD
MSD1R_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD
MSD1R_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD
MSD1R_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD
MSD1W_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD
MSD1W_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD
MSD1W_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD

Dword Atomic Counter Operations

Descriptor
MSD1R_DWAC1 - Dword Atomic Counter Unary with Return Data Operation MSD
MSD1R_DWAC2 - Dword Atomic Counter Binary with Return Data Operation MSD
MSD1W_DWAC1 - Dword Atomic Counter Unary Write Only Operation MSD
MSD1W_DWAC2 - Dword Atomic Counter Binary Write Only Operation MSD

Descriptor
MSD1R_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD
MSD1R_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD
MSD1W_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD
MSD1W_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD

Data Port 2 Message Specific Descriptors

This section contains the message specific descriptors for Data Port 2.

MT_DP2 - Data Port 2 Message Types

A32 and SLM Scaled Read/Write

Message Descriptor
MSD2R_BS - Byte Scaled Read MSD
MSD2W_BS - Byte Scaled Write MSD
MSD2R_US - Scaled Untyped Surface Read MSD
MSD2W_US - Scaled Untyped Surface Write MSD

A64 Scaled Read/Write

Message Descriptor
MSD2R_A64_BS - A64 Byte Scaled Read MSD
MSD2W_A64_BS - A64 Byte Scaled Write MSD
MSD2R_A64_DWS - A64 Dword Scaled Read MSD
MSD2W_A64_DWS - A64 Dword Scaled Write MSD
MSD2R_A64_QWS - A64 Qword Scaled Read MSD
MSD2W_A64_QWS - A64 Qword Scaled Write MSD
MSD2R_A64_US - A64 Scaled Untyped Surface Read MSD
MSD2W_A64_US - A64 Scaled Untyped Surface Write MSD

Read-Only Data Port Message Specific Descriptors

This section contains the message specific descriptors for the Read-Only Data Port .

MT_DP_RO - Read-Only Data Port Message Types

Block Read/Write

Message Descriptor
MSD_CC_DWS - Constant Cache Dword Scattered Read MSD
MSD_CC_OWB - Constant Cache Oword Block Read MSD
MSD_CC_OWUB - Constant Cache Oword Unaligned Block Read MSD

Message Descriptor
MSD_CC_OWDB - Constant Cache Oword Dual Block Read MSD
MSD_SC_OWUB - Sampler Cache Oword Unaligned Block Read MSD
MSD_SC_MB - Sampler Cache Media Block Read MSD

Other Messages

Message Descriptor
MSD_RSI - Read Surface Info MSD

Message Headers

Message Headers provide additional parameters used by the message. When present, the Header is 1 register and specifies a common set of parameters used by most of the data port's messages. The specific combinations supported are described in the Messages section.

Data Port 0 supports various block and scattered operations for the Read/Write data cache. The Data Port 0 Message Header is programmed to provide:

Buffer Base Address	A base address offset used for A32 stateless address models. The default value is 0 when the message header is not present.
Global Offset	A Byte, DWord, or QWord offset that is added to base address offset to address the data operands. The default value is 0 when the message header is not present.
Per Thread Scratch Space	This is used for bounds checking by the data port operation, to guarantee that the address calculation does not go outside of the range expected for this thread. The default value is the maximum value (11) when the message header is not present.

The A64 Read/Write Block operation provides A64 Stateless operations on the Data Port 1 Read/Write data cache. This special Data Port 1 Message Header is programmed to provide:

Block Offset 0 **Block Offset 1**	This specifies the 1 or 2 64-bit addresses used for the A64 stateless address model.
HWord Channel Mode	This controls how channel execution masks are interpreted for HWord operations.

Surface and atomic operations, both untyped and typed, are provided on Data Port 1 for the Read/Write data cache. This Message Header is programmed to provide:

Pixel Sample Mask	This specifies the channel mask for SIMD8 and SIMD16 operations. The default value is 0FFFFh when the message header is not present.
Buffer Base Address	A base address used for SLM and A32 stateless address models. The default value is 0 when the message header is not present.

Scaled untyped surface and byte-aligned operations are provided on Data Port 2 for the Read/Write data cache. This Message Header is programmed to provide:

Pixel Sample Mask	This specifies the channel mask for SIMD8 and SIMD16 operations for surface operations.
Surface Pitch	Scale factor for stateless messages.
Buffer Base Address	A base address used for stateless address models.

Data Port 0 Message Headers

This section contains the message headers for Data Port 0.

Message Header
MH_BTS_GO - Block Message Header
MH_A32_GO - Stateless Block Message Header
MH_A32_HWB - Scratch Hword Block Message Header
MH_IGNORE - Ignored Message Header

Data Port 1 Message Headers

This section contains the message headers for Data Port 1.

Message Header
MH_A64_OWB - A64 Oword Block Message Header
MH_A64_OWDB - A64 Dual Oword Block Message Header
MH_A64_HWB - A64 Hword Block Message Header
MH1_BTS_PSM - Surface Pixel Mask Message Header
MH1_SLM_PSM - SLM Surface Pixel Mask Message Header
MH1_A32_PSM - Stateless Surface Pixel Mask Message Header
MH1_A32 - Stateless Surface Message Header
MH_T - Transpose Message Header
MH_MB - Normal Media Block Message Header
MH_MBPM - Pixel Masked Media Block Message Header
MH_MBBM - Byte Masked Media Block Message Header
MH_MBM - Merged Media Block Message Header

Data Port 2 Message Headers

This section contains the message headers for Data Port 2.

Message Header
MH2_A32 - Stateless Scaled Data Port 2 Message Header
MH2_A32_PSM - Stateless Scaled Data Port 2 Pixel Mask Message Header
MH2_A64 - A64 Scaled Data Port 2 Message Header

Message Address Payloads

The next sections describe all the supported formats of the Address Payloads.

Message Address Payloads are packed into registers based on the address model, surface type, and the SIMD size. The number of registers used by an address payload is 1, 2, 4, or 5.

32 Bit Address Payloads

Address Payload
MAP32b_SIMD4x2 - SIMD4x2 32-Bit Address Payload
MAP32b_SIMD8 - SIMD8 32-Bit Address Payload
MAP32b_SIMD16 - SIMD16 32-Bit Address Payload

64 Bit Address Payloads

Address Payload
MAP64b_SIMD8 - SIMD8 64-Bit Address Payload
MAP64b_SIMD16 - SIMD16 64-Bit Address Payload

32 Bit Untyped Surface Address Payloads

Address Payload
MAP32b_USU_SIMD4x2 - SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload
MAP32b_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 32-Bit Address Payload
MAP32b_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 32-Bit Address Payload
MAP32b_USUV_SIMD4x2 - SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload
MAP32b_USUV_SIMD8 - SIMD8 Untyped STRBUF Surface 32-Bit Address Payload
MAP32b_USUV_SIMD16 - SIMD16 Untyped STRBUF Surface 32-Bit Address Payload

64 Bit Untyped Surface Address Payloads

Address Payload
MAP64b_USU_SIMD4x2 - SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload
MAP64b_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 64-Bit Address Payload
MAP64b_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 64-Bit Address Payload

32 Bit Typed Surface Address Payloads

Address Payload
MAP32b_TS_SIMD4x2 - SIMD4x2 Typed Surface 32-Bit Address Payload
MAP32b_TS_SIMD8 - SIMD8 Typed Surface 32-Bit Address Payload
MAP32b_MSAA_TS_SIMD4x2 - SIMD4x2 MSAA Typed Surface 32-Bit Address Payload
MAP32b_MSAA_TS_SIMD8 - SIMD8 MSAA Typed Surface 32-Bit Address Payload

Address Payload
MAP32b_RSI - Read Surface Info 32-Bit Address Payload

Message Data Payloads

The next sections describe the all the supported formats of Data Payloads.

Message Data Payloads are packed into registers based on the width of the data and on the SIMD size. The supported data payloads are 1, 2, 4, or 8 registers.

Oword Block Data Payloads

Payload
MDP_OW1L - Lower Oword Block Data Payload
MDP_OW1U - Upper Oword Block Data Payload
MDP_OW2 - Oword 2 Block Data Payload
MDP_OW4 - Oword 4 Block Data Payload
MDP_OW8 - Oword 8 Block Data Payload
MDP_OWD1 - Oword 1 Dual Block Data Payload
MDP_OWD4 - Oword 4 Dual Block Data Payload

Hword Block Data Payloads

Payload
MDP_HW1 - Hword 1 Block Data Payload
MDP_HW2 - Hword 2 Block Data Payload
MDP_HW4 - Hword 4 Block Data Payload
MDP_HW8 - Hword 8 Block Data Payload

Dword SIMD Data Payloads

Payload
MDP_DW_SIMD8 - Dword SIMD8 Data Payload
MDP_DW_SIMD16 - Dword SIMD16 Data Payload
MDP_DW_SIMD4x2 - Dword SIMD4x2 Data Payload

Qword SIMD Data Payloads

Payload
MDP_QW_SIMD8 - Qword SIMD8 Data Payload
MDP_QW_SIMD16 - Qword SIMD16 Data Payload

SIMD Atomic Operation Data Payloads

Payload
MDP_AOP8_DW2 - Dword SIMD8 Atomic Operation CMPWR Message Data Payload
MDP_AOP16_DW2 - Dword SIMD16 Atomic Operation CMPWR Message Data Payload
MDP_AOP8_QW1 - Qword SIMD8 Atomic Operation Return Data Message Data Payload
MDP_AOP16_QW1 - Qword SIMD16 Atomic Operation Return Data Message Data Payload
MDP_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload
MDP_AOP16_QW2 - Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload
MDP_A64_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR Message Data Payload
MDP_A64_AOP8_OW2 - Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload
MDP_AOP4x2_DW1 - Dword SIMD4x2 Atomic Operation Message Data Payload
MDP_AOP4x2_DW2 - Dword SIMD4x2 Atomic CMPWR Message Data Payload
MDP_AOP4x2_QW1 - Qword SIMD4x2 Atomic Operation Message Data Payload
MDP_AOP4x2_QW2 - Qword SIMD4x2 Atomic CMPWR8B Message Data Payload
MDP_A64_AOP4x2_QW2 - Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload
MDP_A64_AOP4x2_OW1 - Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload
MDP_A64_AOP4x2_OW2 - Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload

Other Data Payloads

Payload
MDP_RSI - Read Surface Info Data Payload
MDP_TileW_SIMD8 - TileW SIMD8 Data Payload

Common Message Descriptor Controls

Many messages are related and share a common set of control fields. For example, many read and write operations use the same Message Specific Controls in their message descriptor. These common fields are described in the next sections.

Binding Table Index Message Descriptor Control Fields

Field
MDC_BTS - Surface Binding Table Index Message Descriptor Control Field
MDC_STATELESS - Stateless Binding Table Index Message Descriptor Control Field
MDC_BTS_A32 - Surface or Stateless Binding Table Index Message Descriptor Control Field
MDC_BTS_SLM_A32 - Any Binding Table Index Message Descriptor Control Field

Header Present Message Descriptor Control Fields

Field
MDC_MHP - Header Present Message Descriptor Control Field
MDC_MHR - Header Required Message Descriptor Control Field
MDC_MHF - Header Forbidden Message Descriptor Control Field
MDC_A32_MHP - A32 Scaled Header Present Message Descriptor Control Field
MDC_A64_MHP - A64 Scaled Header Present Message Descriptor Control Field

Data Blocks Message Descriptor Control Fields

Field
MDC_DB_OW - Oword Data Blocks Message Descriptor Control Field
MDC_DB_OWD - Oword Dual Data Blocks Message Descriptor Control Field
MDC_DB_HW - Hword Register Blocks Message Descriptor Control Field
MDC_A64_DB_OW - A64 Oword Data Blocks Message Descriptor Control Field
MDC_A64_DB_OWD - A64 Oword Dual Data Blocks Message Descriptor Control Field
MDC_A64_DB_HW - A64 Hword Data Blocks Message Descriptor Control Field
MDC_DS - Data Size Message Descriptor Control Field
MDC_A64_DS - A64 Data Size Message Descriptor Control Field

SIMD Mode Message Descriptor Control Fields

Field
MDC_SM3 - SIMD Mode 3 Message Descriptor Control Field
MDC_SM3S - Subset SIMD Mode 3 Message Descriptor Control Field
MDC_SM2 - SIMD Mode 2 Message Descriptor Control Field
MDC_SM2R - Reversed SIMD Mode 2 Message Descriptor Control Field
MDC_SM2S - Subset SIMD Mode 2 Message Descriptor Control Field
MDC_SG3 - Slot Group 3 Message Descriptor Control Field
MDC_SG2 - Slot Group 2 Message Descriptor Control Field

Atomic Operation Message Descriptor Control Fields

Field
MDC_AOP1 - Atomic Integer Unary Operation Message Descriptor Control Field
MDC_AOP2 - Atomic Integer Binary Operation Message Descriptor Control Field
MDC_AOP3 - Atomic Integer Trinary Operation Message Descriptor Control Field
MDC_AOP3S - Subset Atomic Integer Trinary Operation Message Descriptor Control Field
MDC_FOP2 - Atomic Float Binary Operation Message Descriptor Control Field
MDC_FOP3 - Atomic Float Trinary Operation Message Descriptor Control Field

Other Message Descriptor Control Fields

Field
MDC_IAR - Invalidate After Read Message Descriptor Control Field
MDC_CMODE - Channel Mode Message Descriptor Control Field
MDC_CMASK - Channel Mask Message Descriptor Control Field
MDC_UW_CMASK - Untyped Write Channel Mask Message Descriptor Control Field
MDC_A32_SBSO - A32 Sideband Scale and Offset Enable Message Descriptor Control Field
MDC_A64_SBSO - A64 Sideband Scale and Offset Enable Message Descriptor Control Field
MDC_VLSO - Vertical Line Stride Override Message Descriptor Control Field

Common Message Payload Controls

Some message headers and payloads are related and share a common set of control fields. These common payload fields are described below.

Header Controls

Control
MHC_FFTID - FFTID Message Header Control
MHC_A32_BBA - A32 Buffer Base Address Message Header Control
MHC_A64_CMODE - Hword Channel Mode Message Header Control
MHC_BDIM - Block Dimensions Message Header Control
MHC_MB_CONTROL - Normal Media Block Message Header Control
MHC_MBBM_CONTROL - Byte Masked Media Block Message Header Control
MHC_MBM_CONTROL - Merged Media Block Message Header Control
MHC_MBPM_CONTROL - Pixel Masked Media Block Message Header Control
MHC_PITCH - Surface Pitch Message Header Control
MHC_PSM - Pixel Sample Mask Message Header Control
MHC_PTSS - Per Thread Scratch Space Message Header Control

Address Controls

Control
MACR_32b - SIMD 32-Bit Address Payload Control
MACR_64b - SIMD 64-Bit Address Payload Control
MACD_LOD - LOD Message Address Payload Control
MACR_LOD_SIMD8 - SIMD8 LOD Message Address Payload Control
MACD_MSAA_SN - MSAA Sample Number Message Address Control

Data Controls

Control
MDCR_DW - Dword Data Payload Register
MDCR_QW - Qword Data Payload Register
MDCR_OW - Oword Data Payload Register
MDCD_TileW - TileW SIMD8 Data Control Dword

Pixel Data Port

Cache Agents

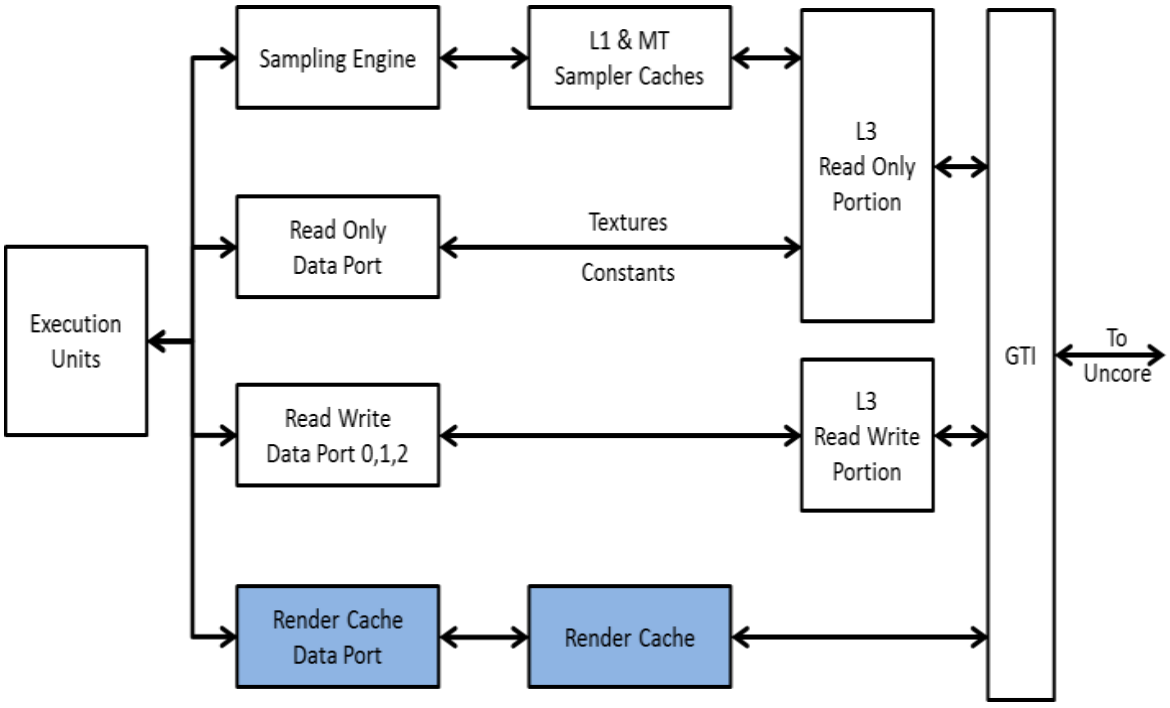
The pixel data port allows read/write accesses to render targets. Render target is cached in Render Color Cache (RCC). Auxiliary Surafce (a.k.a Control Surface) i.e. MCS buffer corresponding to a render target is cached in MSC. Messages on O-bus with Shared Function ID = Render Target (RCC) are routed to Pixel Dataport.

Shared Functions - Render Target

The Render Cache Data Port provides memory accesses for the Gen subsystem for the Render Cache. The render cache is a read/write cache that supports 3D render target surfaces.

The diagram below shows how the Render Cache Data Port connects with the caches and memory subsystem. The execution units and sampling engine are shown for clarity.

Data Port Render Cache Connection to Memory



Render Target

The final results of pixel shaders are written to either UAV surfaces or to Render target surfaces. Render targets support a large set of surface formats (refer to *Render Target Surfaces* for details) with hardware conversion from the format delivered by the thread. The render target message processing in HW includes format conversion, alpha-test, alpha-to-coverage, depth/stencil tests, blending and logop depending on various states.

The render target read/write messages are specifically for the use of pixel shader threads spawned by the Pixel Shader Dispatch(PSD), and may not be used by any other thread types.

Render Target Surfaces

This section contains information on render target surface types and surface formats.

Render Target Surface Types

All surface types, except SURFTYPE_STRBUF, are allowed.

For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.

For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a Render Target Array Index is included in the input message to provide an additional coordinate. The Render Target Array Index must be zero for SURFTYPE_BUFFER.

The surface format is restricted to the set supported as render target. If source/dest color blend is enabled on Render Target Write message, the surface format is further restricted to the set supported as alpha blend render target.

Cannot be used on a surface in field mode (Vertical Line Stride = 1).

Render Target Write Surface Formats

The following table indicates the surface formats supported by the Render Target Write message. All Render Target formats supported for Render Target Writes are also supported for Render Target Reads, except YCRCB formats.

The table lists the Surface Formats supported for reading and/or writing of Render Targets. For blending operations, the named Projects support color blending on the listed Surface Formats.

Surface Formats for Render Target Messages

Surface Format Name	Color Blend Support	Message Type Support	Lossless Compression Support
R32G32B32A32_FLOAT	All	Read/Write	
R32G32B32A32_SINT	*Not Supported*	Read/Write	
R32G32B32A32_UINT	*Not Supported*	Read/Write	
R16G16B16A16_UNORM	All	Read/Write	
R16G16B16A16_SNORM	All	Read/Write	
R16G16B16A16_SINT	*Not Supported*	Read/Write	
R16G16B16A16_UINT	*Not Supported*	Read/Write	
R16G16B16A16_FLOAT	All	Read/Write	
R32G32_FLOAT	All	Read/Write	
R32G32_SINT	*Not Supported*	Read/Write	
R32G32_UINT	*Not Supported*	Read/Write	
R16G16B16X16_FLOAT	[CHV]	Read/Write	
B8G8R8A8_UNORM	All	Read/Write	
R8G8B8A8_UNORM	All	Read/Write	
R8G8B8A8_SNORM	All	Read/Write	
R8G8B8A8_SINT	*Not Supported*	Read/Write	
R8G8B8A8_UINT	*Not Supported*	Read/Write	
R16G16_UNORM	All	Read/Write	
R16G16_SNORM	All	Read/Write	
R16G16_SINT	*Not Supported*	Read/Write	
R16G16_UINT	*Not Supported*	Read/Write	
R16G16_FLOAT	All	Read/Write	
R32_SINT	*Not Supported*	Read/Write	
R32_UINT	*Not Supported*	Read/Write	
R32_FLOAT	All	Read/Write	
B8G8R8X8_UNORM		Read/Write	
B5G6R5_UNORM	All	Read/Write	
B5G6R5_UNORM_SRGB	All	Read/Write	
B5G5R5A1_UNORM	All	Read/Write	
B5G5R5A1_UNORM_SRGB	All	Read/Write	
B4G4R4A4_UNORM	All	Read/Write	
B4G4R4A4_UNORM_SRGB	All	Read/Write	
R8G8_UNORM	All	Read/Write	

Surface Format Name	Color Blend Support	Message Type Support	Lossless Compression Support
R8G8_SNORM	All	Read/Write	
R8G8_SINT	*Not Supported*	Read/Write	
R8G8_UINT	*Not Supported*	Read/Write	
R16_UNORM	All	Read/Write	
R16_SNORM	All	Read/Write	
R16_SINT	*Not Supported*	Read/Write	
R16_UINT	*Not Supported*	Read/Write	
R16_FLOAT	All	Read/Write	
B5G5R5X1_UNORM	All	Read/Write	
B5G5R5X1_UNORM_SRGB	All	Read/Write	
A1B5G5R5_UNORM	[CHV]	Read/Write	
A4B4G4R4_UNORM	[CHV]	Read/Write	
R8_UNORM	All	Read/Write	
R8_SNORM	All	Read/Write	
R8_SINT	*Not Supported*	Read/Write	
R8_UINT	*Not Supported*	Read/Write	
A8_UNORM	All	Read/Write	
YCRCB_NORMAL	*Not Supported*	Write	
YCRCB_SWAPUVY	*Not Supported*	Write	
YCRCB_SWAPUV	*Not Supported*	Write	
YCRCB_SWAPY	*Not Supported*	Write	

Subspan/Pixel to Slot Mapping

The mapping of subspans, pixels, and samples to slots in the pixel shader dispatch depends on the number of samples and message size.

Pixels are numbered as follows within a subspan:

0 = upper left

1 = upper right

2 = lower left

3 = lower right

sspi = Starting Sample Pair Index (from the message header)

Slot Mapping

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0] Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0] Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1] Slot[19:16] = Subspan[2].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[2].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[3].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[3].Pixel[3:0].Sample[1]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[1].Pixel[3:0].Sample[2] Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]
	8X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4] Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5] Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6] Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]
SIMD16	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]
	8X	Dispatch[i]: (i=0, 2) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]
	16x	Dispatch[i]: (i=0, 2, 4, 6) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD8	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]
	4X	Dispatch[i]: (i=0..1) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]
	8X	Dispatch[i]: (i=0, 1, 2, 3) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]
	16x	Dispatch[i]: (i=0, 1, 2, 3, 4, 5, 6, 7) SSPI = i Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]

Programming Note	
Context:	Render Target Surfaces
• When SIMD32 or SIMD16 PS threads send render target writes with multiple SIMD8 and SIMD16 messages, the following must hold: • All the slots (as described above) must have a corresponding render target write irrespective of the slot's validity. A slot is considered valid when at least one sample is enabled. For example, a SIMD16 PS thread must send two SIMD8 render target writes to cover all the slots.	

Message Common Control Fields

The following data tables describe common control fields used in the Render Target message descriptors.

MDC_RT_SGS - Slot Group Select Render Cache Message Descriptor Control Field

Render Target Messages

The message operations on Render Cache Data Port use the same interface as the other data port messages. The message's Descriptor is sent to the target data port on a message sideband bus, and the other parts are sequentially transmitted between the data port and EU registers across the OBUS. The total number of registers sent and received by the data port is provided to the data port on the message sideband bus.

Descriptor	Describes the Message Type, the Message Specific Controls, and whether a Message Header is present. The Message Descriptor and the destination Data Port (SFID) are encoded as part of the *send* instruction.
Header	When present, provides additional Message specific controls. The message header is optional for some Render Target messages. When present, it is a 2-register payload.
Source Payload	Provides source data for write operations operations.
Writeback Payload	Returns result data for read operations operations.

The next sections describe all of the supported formats for the Source and Writeback Data Payloads, the Message Descriptors, and their Message Headers.

Message Summary tables describe the parameters and characteristics of the Render Target data port messages, following the notational conventions used in Message Formats.

Render Target Write Message

Addr Align	Data Width	R/W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	W	BTS	1D, 2D, 3D, CUBE, BUFFER	8, 16	1	(Surface[U, V, R, LOD])→SurfaceFormat[Slot]	Surface	RTPSM

This message writes 8 or 16 pixels to a render target. Depending on parameters contained in the message and state, it may also perform a depth and stencil buffer write and/or a render target read for a color blend operation. Additional operations enabled in the Color Calculator state are also initiated as a result of issuing this message (depth test, alpha test, logic ops, etc.). This message is intended only for use by pixel shader kernels for writing results to render targets.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	{Opt} MH_RT	{Forbidden}	MDP_RTW_8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_ZM	{Forbidden}	MDP_RTW_ZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_Z	{Forbidden}	MDP_RTW_Z8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_M	{Forbidden}	MDP_RTW_M8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	{Opt} MH_RT	{Forbidden}	MDP_RTW_8DS	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_ZM	{Forbidden}	MDP_RTW_ZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_Z	{Forbidden}	MDP_RTW_Z8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_M	{Forbidden}	MDP_RTW_M8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	{Opt} MH_RT	{Forbidden}	MDP_RTW_8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZMA	{Forbidden}	MDP_RTW_ZMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZA	{Forbidden}	MDP_RTW_ZA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_MA	{Forbidden}	MDP_RTW_MA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZM	{Forbidden}	MDP_RTW_ZM8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_Z	{Forbidden}	MDP_RTW_Z8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_A	{Forbidden}	MDP_RTW_A8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_M	{Forbidden}	MDP_RTW_M8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	{Opt} MH_RT	{Forbidden}	MDP_RTW_S8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_ZM	{Forbidden}	MDP_RTW_SZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_Z	{Forbidden}	MDP_RTW_SZ8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_LO8DS	MH_RT_M	{Forbidden}	MDP_RTW_SM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	{Opt} MH_RT	{Forbidden}	MDP_RTW_S8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_ZM	{Forbidden}	MDP_RTW_SZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_Z	{Forbidden}	MDP_RTW_SZ8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_HI8DS	MH_RT_M	{Forbidden}	MDP_RTW_SM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	{Opt} MH_RT	{Forbidden}	MDP_RTW_S8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZMA	{Forbidden}	MDP_RTW_SZMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZA	{Forbidden}	MDP_RTW_SZA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_MA	{Forbidden}	MDP_RTW_SMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_ZM	{Forbidden}	MDP_RTW_SZM8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_Z	{Forbidden}	MDP_RTW_SZ8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_A	{Forbidden}	MDP_RTW_SA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTW_SIMD8	MH_RT_M	{Forbidden}	MDP_RTW_SM8	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	{Opt} MH_RT	{Forbidden}	MDP_RTW_16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_ZMA	{Forbidden}	MDP_RTW_ZMA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_ZA	{Forbidden}	MDP_RTW_ZA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_MA	{Forbidden}	MDP_RTW_MA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_ZM	{Forbidden}	MDP_RTW_ZM16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_Z	{Forbidden}	MDP_RTW_Z16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_A	{Forbidden}	MDP_RTW_A16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_SIMD16	MH_RT_M	{Forbidden}	MDP_RTW_M16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_REP16	{Opt} MH_RT	{Forbidden}	MDP_RTW_16REP	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTW_REP16	MH_RT_M	{Forbidden}	MDP_RTW_M16REP	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	{Opt} MH_RT	{Forbidden}	MDP_RTWH_8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_ZM	{Forbidden}	MDP_RTWH_ZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_Z	{Forbidden}	MDP_RTWH_Z8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_M	{Forbidden}	MDP_RTWH_M8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	{Opt} MH_RT	{Forbidden}	MDP_RTWH_8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_ZM	{Forbidden}	MDP_RTWH_ZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_Z	{Forbidden}	MDP_RTWH_Z8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_M	{Forbidden}	MDP_RTWH_M8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	{Opt} MH_RT	{Forbidden}	MDP_RTWH_8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZMA	{Forbidden}	MDP_RTWH_ZMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZA	{Forbidden}	MDP_RTWH_ZA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_MA	{Forbidden}	MDP_RTWH_MA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZM	{Forbidden}	MDP_RTWH_ZM8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_Z	{Forbidden}	MDP_RTWH_Z8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_A	{Forbidden}	MDP_RTWH_A8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_M	{Forbidden}	MDP_RTWH_M8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	{Opt} MH_RT	{Forbidden}	MDP_RTWH_S8DS	{Forbidden}

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_ZM	{Forbidden}	MDP_RTWH_SZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_Z	{Forbidden}	MDP_RTWH_SZ8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_LO8DS	MH_RT_M	{Forbidden}	MDP_RTWH_SM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	{Opt} MH_RT	{Forbidden}	MDP_RTWH_S8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_ZM	{Forbidden}	MDP_RTWH_SZM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_Z	{Forbidden}	MDP_RTWH_SZ8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_HI8DS	MH_RT_M	{Forbidden}	MDP_RTWH_SM8DS	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	{Opt} MH_RT	{Forbidden}	MDP_RTWH_S8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZMA	{Forbidden}	MDP_RTWH_SZMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZA	{Forbidden}	MDP_RTWH_SZA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_MA	{Forbidden}	MDP_RTWH_SMA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_ZM	{Forbidden}	MDP_RTWH_SZM8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_Z	{Forbidden}	MDP_RTWH_SZ8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_A	{Forbidden}	MDP_RTWH_SA8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD8	1	MSD_RTWH_SIMD8	MH_RT_M	{Forbidden}	MDP_RTWH_SM8	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	{Opt} MH_RT	{Forbidden}	MDP_RTWH_16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_ZMA	{Forbidden}	MDP_RTWH_ZMA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_ZA	{Forbidden}	MDP_RTWH_ZA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_MA	{Forbidden}	MDP_RTWH_MA16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_ZM	{Forbidden}	MDP_RTWH_ZM16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_Z	{Forbidden}	MDP_RTWH_Z16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_A	{Forbidden}	MDP_RTWH_A16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_SIMD16	MH_RT_M	{Forbidden}	MDP_RTWH_M16	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_REP16	{Opt} MH_RT	{Forbidden}	MDP_RTWH_16REP	{Forbidden}
W	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD16	1	MSD_RTWH_REP16	MH_RT_M	{Forbidden}	MDP_RTWH_M16REP	{Forbidden}

The sample for a pixel is killed (not written to the render target or depth buffer) if the corresponding oMask bit is zero. Bits in oMask larger than the number of multisamples are ignored.

The color payload is included if the Message Type is SIMD8 single source or SIMD8 Image Write.

The SIMD16 Replicated Data Color payload is included if the Message Type specifies single source message with replicated data. One set of R/G/B/A data is included in the message, and this data is replicated to all 16 pixels. The Replicated SIMD16 message is legal with color data and oMask, but the registers for depth, stencil, and antialias alpha data cannot be included with this message, and the corresponding bits in the message header must indicate that these registers are not present.

The following table enumerates the order that the data payload is packed in registers: Source 0 Alpha (s0A), Output Mask (oM), Pixel components (R, G, B, A), Source Depth (sZ), and Output Stencil (oS).

Summary of Data Payload Register Order for Render Target Write Messages

Message Type	Present?				Message Register												
	S0A	oM	sZ	oS	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
SIMD16	0	0	0	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A					
SIMD16	1	0	0	0	1/0s0A	3/2s0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A			
SIMD16	0	0	1	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ			
SIMD16	1	0	1	0	1/0s0A	3/2s0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ	
SIMD16	0	1	0	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A				
SIMD16	1	1	0	0	1/0soA	3/2s0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A		
SIMD16	0	1	1	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ		
SIMD16	1	1	1	0	1/0s0A	3/2s0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0sZ	3/2sZ
REP16	0	0	0	0	RGBA												
LO8DS	0	0	0	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A					
LO8DS	0	0	1	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ				
LO8DS	0	1	0	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A				
LO8DS	0	1	1	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ			
HI8DS	0	0	0	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A					
HI8DS	0	0	1	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ				
HI8DS	0	1	0	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A				
HI8DS	0	1	1	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ			
SIMD8	0	0	0	0	R	G	B	A									
SIMD8	1	0	0	0	s0A	R	G	B	A								
SIMD8	0	0	1	0	R	G	B	A	sZ								
SIMD8	1	0	1	0	s0A	R	G	B	A	sZ							
SIMD8	0	1	0	0	oM	R	G	B	A								
SIMD8	1	1	0	0	s0A	oM	R	G	B	A							
SIMD8	0	1	1	0	oM	R	G	B	A	sZ							
SIMD8	1	1	1	0	s0A	oM	R	G	B	A	sZ						
LO8DS	0	0	0	1	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0oS				
LO8DS	0	0	1	1	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ	1/0OoS			
LO8DS	0	1	0	1	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0oS			
LO8DS	0	1	1	1	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ	1/0oS		
HI8DS	0	0	0	1	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2oS				
HI8DS	0	0	1	1	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ	3/2oS			
HI8DS	0	1	0	1	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2oS			
HI8DS	0	1	1	1	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ	3/2oS		
SIMD8	0	0	0	1	R	G	B	A	oS								
SIMD8	1	0	0	1	s0A	R	G	B	A	oS							

SIMD8	0	0	1	1	R	G	B	A	sZ	oS							
SIMD8	1	0	1	1	s0A	R	G	B	A	sZ	oS						
SIMD8	0	1	0	1	oM	R	G	B	A	oS							
SIMD8	1	1	0	1	s0A	oM	R	G	B	A	oS						
SIMD8	0	1	1	1	oM	R	G	B	A	sZ	oS						
SIMD8	1	1	1	1	s0A	oM	R	G	B	A	sZ	oS					
HP SIMD16	0	0	0	0	R	G	B	A									
HP SIMD16	1	0	0	0	s0A	R	G	B	A								
HP SIMD16	0	0	1	0	R	G	B	A	sZ								
HP SIMD16	1	0	1	0	s0A	R	G	B	A	sZ							
HP SIMD16	0	1	0	0	oM	R	G	B	A								
HP SIMD16	1	1	0	0	s0A	oM	R	G	B	A							
HP SIMD16	0	1	1	0	oM	R	G	B	A	sZ							
HP SIMD16	1	1	1	0	s0A	oM	R	G	B	A	sZ						
HP REP16	0	0	0	0	RGBA												
HP LO8DS	0	0	0	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A					
HP LO8DS	0	0	1	0	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ				
HP LO8DS	0	1	0	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A				
HP LO8DS	0	1	1	0	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ			
HP HI8DS	0	0	0	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A					
HP HI8DS	0	0	1	0	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ				
HP HI8DS	0	1	0	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A				
HP HI8DS	0	1	1	0	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ			
HP SIMD8	0	0	0	0	R	G	B	A									
HP SIMD8	1	0	0	0	s0A	R	G	B	A								
HP SIMD8	0	0	1	0	R	G	B	A	sZ								
HP SIMD8	1	0	1	0	s0A	R	G	B	A	sZ							
HP SIMD8	0	1	0	0	oM	R	G	B	A								
HP SIMD8	1	1	0	0	s0A	oM	R	G	B	A							
HP SIMD8	0	1	1	0	oM	R	G	B	A	sZ							
HP SIMD8	1	1	1	0	s0A	oM	R	G	B	A	sZ						
HP LO8DS	0	0	0	1	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0oS				
HP LO8DS	0	0	1	1	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ	1/0OoS			
HP LO8DS	0	1	0	1	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0oS			
HP LO8DS	0	1	1	1	oM	1/0s0R	1/0s0G	1/0s0B	1/0s0A	1/0s1R	1/0s1G	1/0s1B	1/0s1A	1/0sZ	1/0oS		
HP HI8DS	0	0	0	1	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2oS				
HP HI8DS	0	0	1	1	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ	3/2oS			
HP HI8DS	0	1	0	1	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2oS			
HP HI8DS	0	1	1	1	oM	3/2s0R	3/2s0G	3/2s0B	3/2s0A	3/2s1R	3/2s1G	3/2s1B	3/2s1A	3/2sZ	3/2oS		
HP SIMD8	0	0	0	1	R	G	B	A	oS								
HP SIMD8	1	0	0	1	s0A	R	G	B	A	oS							
HP SIMD8	0	0	1	1	R	G	B	A	sZ	oS							
HP SIMD8	1	0	1	1	s0A	R	G	B	A	sZ	oS						
HP SIMD8	0	1	0	1	oM	R	G	B	A	oS							
HP SIMD8	1	1	0	1	s0A	oM	R	G	B	A	oS						
HP SIMD8	0	1	1	1	oM	R	G	B	A	sZ	oS						
HP SIMD8	1	1	1	1	s0A	oM	R	G	B	A	sZ	oS					

Programming Note	
Context:	Render Target Write Message
Typically, the last message in a pixel shader is a Render Target Write message, with EOT set.	

Programming Note	
Context:	Render Target Write Message
• The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG. • If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same Slot Group Select setting. • Output Stencil is not supported with SIMD16 Render Target Write Messages.	

Programming Note	
Context:	Render Target Writes in Multirate Shaders
• In multirate shaders, render target writes that can potentially kill pixels (via oMask, oDepth, or oStencil) cannot modify the oMask, oDepth, or oStencil value after writing to a render target. For instance if a pixel-rate render target is written, an associated sample-rate render target cannot modify these values or undefined behavior can occur. • In multirate shaders that write to render targets in multiple stages, render target 0 must be written to before any other render targets..	

Replicate Data

The replicate data render target message is used for clearing Render Target. This message performs better than the other messages due to its smaller message length. This message does not support depth, stencil, or antialias alpha data being sent with it.

The pixel scoreboard bits corresponding to the dispatched pixel mask are cleared only if the Last Render Target Select bit is set in the message descriptor.

Programming Note	
Context:	Replicate Data Render Target Write Message
Replicate Data Render Target Write message is not supported with AA alpha i.e. when primitive has Anti-Aliased alpha enabled e.g. AA lines and points.	

Single Source

The “normal” render target messages are single source. There are two forms, SIMD16 and SIMD8, intended for the equivalent-sized pixel shader threads. A single source (4 channels) is delivered for each of the 16 or 8 pixels in the message payload. Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

The pixel scoreboard bits corresponding to the dispatched pixel mask (or half of the mask in the case of SIMD8 messages) are cleared only if the Last Render Target Select bit is set in the message descriptor. However, if Last Render Target Select is set, the message still causes pixel scoreboard clear and depth/stencil buffer updates if enabled.

Dual Source

The dual source render target messages only have SIMD8 forms due to maximum message length limitations. SIMD16 pixel shaders must send two of these messages to cover all of the pixels. Each message contains two colors (4 channels each) for each pixel in the message payload. In addition to the first source, the second source can be selected as a blend factor (BLENDFACTOR_*_SRC1_* options in the blend factor fields of COLOR_CALC_STATE or BLEND_STATE). Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

For SIMD16 PS thread with two output colors must send messages in the followng sequence for each RT: SIMD8 dual source RTW message (low); SIMD8 dual source RTW message (high); SIMD16 single src RTW message with second color.

Render Target Read Message

Addr Align	Data Width	R / W	Address Model	Surface Type	SIMD Slots	Data Elements	SIMD Address Calculation	Bounds Check	Execution Mask
DW	DW	R	BTS	1D, 2D, 3D, CUBE, BUFFER	8, 16	4	(Surface[U, V, R, LOD])→DW[Chan]	Surface	RTPSM

This message takes 8 or 16 pixels for reads to a render target. This message is intended only for use by pixel shader kernels for reading data from render targets.

R/W	Address Model	Surface Type	SIMD Slots	Data Elements	Message Specific Descriptor	Message Header	Address Payload	Source Payload	Writeback Payload
R	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD 8	{4}	MSD_RTR_SIMD8	MH_RT	{Forbidden}	{Forbidden}	{4} MDP_DW_SIMD16
R	BTS	1D, 2D, 3D, CUBE, BUFFER	SIMD 16	{4}	MSD_RTR_SIMD16	MH_RT	{Forbidden}	{Forbidden}	{4} MDP_DW_SIMD8

A SIMD8 writeback message consists of 4 destination registers. A SIMD16 writeback message consists of 8 destination registers.

Restrictions:

Must not have End-of-Thread bit set in message.

This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message causes undefined behavior.

Render Target read on an MSRT is supported via per-sample pixel shader.

Programming Note	
Context:	Render Target Read Message
For 1d surfaces Render Target Read Messages are not supported.	

Message-Specific Descriptors

The Render Cache data port has only 2 message types: Render Target Write and Render Target Read. The Render Target Write message type has 5 subtype operations. The Render Target Read message type has 2 subtype operations. The specific message descriptors for these messages are below.

Message Headers

The render target messages use a two-register message header.

If the header is not present, the behavior is as if the message was sent with most fields set to the same value that was delivered in R0 and R1 on the pixel shader thread dispatch. The following fields, which are not delivered in the pixel shader dispatch, behave as if they are set to zero:

Render Target Index

Source0 Alpha Present to Render Target

Additionally, **Render Target Read** messages must include a header.

MH_RT - Render Target Message Header

Message Data Payloads

The data payloads for Render Target Read messages depend on the SIMD message subtype and the channel enables in the message descriptor. The writeback data payloads are 4 channels of red, green, blue, and alpha data. Each channel is one or two message registers laid out in the standard 8 or 16 DWord format, depending whether it is a SIMD8 or SIMD16 data operation.

The data payloads for Render Target Write messages depend on the Message Subtype from the message descriptor, and the settings of the oMask, Source 0 Alpha, and Source Depth presence control bits in the message header. The following data tables list all the valid source payloads combinations for the Render Target Write message.

The half precision Render Target Write messages have data payloads that can pack a full SIMD16 payload into 1 register instead of two. The half-precision packed format is used for RGBA and Source 0 Alpha, but Source Depth data payload is always supplied in full precision.

Render Target Data Payloads

This section contains various registers for the Render Target Data Payloads.

MDP_RTW_16 - SIMD16 Render Target Data Payload

MDP_RTW_ZMA16 - SZ OM S0A SIMD16 Render Target Data Payload

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

MDP_RTW_MA16 - OM S0A SIMD16 Render Target Data Payload

MDP_RTW_ZM16 - SZ OM SIMD16 Render Target Data Payload

MDP_RTW_Z16 - SZ SIMD16 Render Target Data Payload

MDP_RTW_A16 - S0A SIMD16 Render Target Data Payload

MDP_RTW_M16 - OM SIMD16 Render Target Data Payload

MDP_RTW_16REP - Replicated SIMD16 Render Target Data Payload

MDP_RTW_M16REP - OM Replicated SIMD16 Render Target Data Payload

MDP_RTW_8 - SIMD8 Render Target Data Payload

MDP_RTW_ZMA8 - SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTW_ZM8 - SZ OM SIMD8 Render Target Data Payload

MDP_RTW_ZA8 - SZ S0A SIMD8 Render Target Data Payload

MDP_RTW_MA8 - OM S0A SIMD8 Render Target Data Payload

MDP_RTW_Z8 - SZ SIMD8 Render Target Data Payload

MDP_RTW_A8 - S0A SIMD8 Render Target Data Payload

MDP_RTW_M8 - OM SIMD8 Render Target Data Payload

MDP_RTW_S8 - OS SIMD8 Render Target Data Payload

MDP_RTW_SZMA8 - OS SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTW_SZM8 - OS SZ OM SIMD8 Render Target Data Payload

MDP_RTW_SZA8 - OS SZ S0A SIMD8 Render Target Data Payload

MDP_RTW_SMA8 - OS OM S0A SIMD8 Render Target Data Payload

MDP_RTW_SZ8 - OS SZ SIMD8 Render Target Data Payload

MDP_RTW_SA8 - OS S0A SIMD8 Render Target Data Payload

MDP_RTW_SM8 - OS OM SIMD8 Render Target Data Payload

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload

MDP_RTW_ZM8DS - SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_Z8DS - SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_S8DS - OS SIMD8 Dual Source Render Target Data Payload

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_SZM8DS - OS SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_SZ8DS - OS SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_16 - Half Precision SIMD16 Render Target Data Payload

MDP_RTW_SZ8DS - OS SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_ZM16 - Half Precision SZ OM SIMD16 Render Target Data Payload

MDP_RTWH_ZMA16 - Half Precision SZ OM S0A SIMD16 Render Target Data Payload

MDP_RTWH_ZA16 - Half Precision SZ S0A SIMD16 Render Target Data Payload

MDP_RTWH_MA16 - Half Precision OM S0A SIMD16 Render Target Data Payload

MDP_RTWH_Z16 - Half Precision SZ SIMD16 Render Target Data Payload

MDP_RTWH_A16 - Half Precision S0A SIMD16 Render Target Data Payload

MDP_RTWH_M16 - Half Precision OM SIMD16 Render Target Data Payload

MDP_RTWH_16REP - Half Precision Replicated SIMD16 Render Target Data Payload

MDP_RTWH_M16REP - Half Precision OM Replicated SIMD16 Render Target Data Payload

MDP_RTWH_8 - Half Precision SIMD8 Render Target Data Payload

MDP_RTWH_ZMA8 - Half Precision SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_ZM8 - Half Precision SZ OM SIMD8 Render Target Data Payload

MDP_RTWH_ZA8 - Half Precision SZ S0A SIMD8 Render Target Data Payload

MDP_RTWH_MA8 - Half Precision OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_Z8 - Half Precision SZ SIMD8 Render Target Data Payload

MDP_RTWH_A8 - Half Precision S0A SIMD8 Render Target Data Payload

MDP_RTWH_M8 - Half Precision OM SIMD8 Render Target Data Payload

MDP_RTWH_S8 - Half Precision OS SIMD8 Render Target Data Payload

MDP_RTWH_SZMA8 - Half Precision OS SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_SZM8 - Half Precision OS SZ OM SIMD8 Render Target Data Payload

MDP_RTWH_SZA8 - Half Precision OS SZ S0A SIMD8 Render Target Data Payload

MDP_RTWH_SMA8 - Half Precision OS OM S0A SIMD8 Render Target Data Payload

MDP_RTWH_SZ8 - Half Precision OS SZ SIMD8 Render Target Data Payload

MDP_RTWH_SA8 - Half Precision OS S0A SIMD8 Render Target Data Payload

MDP_RTWH_SM8 - Half Precision OS OM SIMD8 Render Target Data Payload

MDP_RTWH_8DS - Half Precision SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_ZM8 - Half Precision SZ OM SIMD8 Render Target Data Payload

MDP_RTWH_M8DS - Half Precision OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_S8DS - Half Precision OS SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SZM8DS - Half Precision OS SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SZ8DS - Half Precision OS SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTWH_SM8DS - Half Precision OS OM SIMD8 Dual Source Render Target Data Payload

Render Target

The final results of pixel shaders are written to either UAV surfaces or to Render target surfaces. Render targets support a large set of surface formats (refer to *Render Target Surfaces* for details) with hardware conversion from the format delivered by the thread. The render target message processing in HW includes format conversion, alpha-test, alpha-to-coverage, depth/stencil tests, blending and logop depending on various states.

The render target read/write messages are specifically for the use of pixel shader threads spawned by the Pixel Shader Dispatch(PSD), and may not be used by any other thread types.

Multiple Render Targets (MRT)

Pixel Shader can output more than one color. Each color is accessed with a separate Render Target Write message, each with a different surface indicated (different binding table index). In this Multiple Render Target (MRT) case, depth buffer and stencil buffer are updated only by the message(s) to the last render target, indicated by the Last Render Target Select bit set to clear the pixel scoreboard bits.

MRT is not supported when one or more RTs have these surface formats: YCRCB_SWAPUVY, YCRCB_SWAPUV, YCRCB_SWAPY, or YCRCB_NORMAL.

Flushing the Render Cache

The render cache can be flushed via PIPE_CONTROL command with “Render Target Cache Flush” bit set. Typically, end of pipe synchronization that requires render targets to be made visible to Sampler or CPU or Display should use this mechanism to flush render cache. With the Render Target Cache Flush feature, some cases do require flushing render cache as mentioned in the respective feature’s sections.

End of Thread Usage

Render Target data port messages may not have the End of Thread bit set in the message descriptor other than the following exceptions:

The Render Target Write message may have End of Thread set for pixel shader threads dispatched by the windower in non-contiguous dispatch mode.

Execution Mask

For Render Target Write messages, either the message header **Pixel Sample Mask** or the channel execution mask is used to control the pixel write operations. If the Message Header is present, then the

Pixel Sample Mask is used and the channel execution mask is ignored. If the Message Header is not present, then the channel execution mask is used in place of the **Pixel Sample Mask**.

For Render Target Read messages, the channel execution mask is used to control GRF write operations. The message returns zeroes for samples not covered by current pixel coverage mask or samples outside of MSAA index range.

Bounds Checking

Read and write accesses to pixels outside of the surface are considered out-of-bounds. However, if only the **Render Target Array Index** is out of bounds and the rest of the access is in-bounds, then the index is set to zero and the read or write surface access is considered in-bounds.

Writes that are out-of-bounds are suppressed and do not modify memory contents. If an in-bounds pixel component is missing due to the Surface Format, the write is suppressed. If an in-bounds pixel component is masked off (Pixel Sample Mask bit is zero), the write is suppressed.

Reads that are out-of-bounds are suppressed and return zero for all components. If an in-bounds pixel component is missing due to the Surface Format, a zero is returned for RGB and a 1.0 is returned for Alpha. A surface read is performed on any in-bounds pixel where the component is present in the Surface Format, regardless of whether it is masked off (Pixel Sample Mask bit is zero). All render target reads, in-bounds or out-of-bounds, modify the GRF.

Shared Functions Pixel Interpolater

The Pixel Interpolator provides barycentric parameters at various offsets relative to the pixel location. These barycentric parameters are in the same format and layout as those received in the pixel shader dispatch. Please refer to the “Windower” chapter in the “3D Pipeline” volume for more details on barycentric parameters.

Barycentric parameters delivered in the pixel shader payload are at pre-defined positions based on **Barycentric Interpolation Mode** bits selected in 3DSTATE_WM. The pixel interpolator allows barycentric parameters to be computed at additional locations.

Messages

The following is the message definition for the Pixel Interpolator shared function.

Programming Note	
Context:	Messages
Pixel Interpolator messages can only be delivered by pixel shader kernels.	

Execution Mask. Each bit in the execution mask enables the corresponding slot's barycentric parameter return to the destination registers.

Initiating Message

Message Descriptor

Bits	Description
19	**Header Present:** Specifies whether the message includes a header phase. Must be zero for all *Pixel Interpolator* messages. Format = Enable
18:17	Ignored
16	**SIMD Mode.** Specifies the SIMD mode of the message being sent. Format = U1 0: SIMD8 mode 1: SIMD16 mode
15	Ignored
14	**Interpolation Mode.** Specifies which interpolation mode is used. Format = U1 0: Perspective Interpolation 1: Linear Interpolation **Programming Note** **Context:** Message Descriptor This field cannot be set to "Linear Interpolation" unless **Non-Perspective Barycentric Enable** in 3DSTATE_CLIP is enabled.

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>13:12</td><td>Message Type. Specifies the type of message being sent when pixel-rate evaluation requested.
Format = U2
0: Per Message Offset (eval_snapped with immediate offset)
1: Sample Position Offset (eval_sindex)
2: Centroid Position Offset (eval_centroid)
3: Per Slot Offset (eval_snapped with register offset)
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><th>Context:</th><td>Message Descriptor</td></tr>
<tr><td colspan="2">When Message Type is Sample Position, requesting an attribute at sample index beyond the range defined by the Forced Sample Count (aka NUM_RASTSAMPLES) is illegal.</td></tr>
</table></td></tr>
<tr><td>11</td><td>Slot Group Select. This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
Bypassed data includes the X/Y addresses and centroid position. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed.
0: SLOTGRP_LO: Choose bypassed data for slots 15:0.
1: SLOTGRP_HI: Choose bypassed data for slots 31:16.
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><th>Context:</th><td>Message Descriptor</td></tr>
<tr><td colspan="2">This field must be set to SLOTGRP_LO for SIMD8 messages. SIMD8 messages always use bypassed data for slots 7:0.</td></tr>
</table></td></tr>
<tr><td>10:8</td><td>Ignored</td></tr>
<tr><td>7:0</td><td>Message Specific Control. Refer to the sections below for the definition of these bits based on Message Type.</td></tr>
</table>

"Per Message Offset" Message Descriptor

Bit	Description
7:4	**Per Message Y Pixel Offset** Specifies the Y Pixel Offset that applies to all slots. Format = S4 2's complement representing units of 1/16 pixel. Range = [-8/16, +7/16]
3:0	**Per Message X Pixel Offset** Specifies the X Pixel Offset that applies to all slots. Format = S4 2's complement representing units of 1/16 pixel. Range = [-8/16, +7/16]

"Sample Position Offset" Message Descriptor

Bits	Description
7:4	**Sample Index** Specifies the sample index that applies to all slots. Sample Index must not exceed the value of NUM_RASTSAMPLES when NUM_RASTSAMPLES > 1. From API, perspective, Forced Sample Count Defines the maximum allowable index in this message. Format = U4 Range [0, 15]
3:0	Ignored

"Centroid Position" and "Per Slot Offset" Message Descriptor

Bit	Description
7:0	Ignored

Message Payload for Most Messages

This message payload applies to the following message types:

- Per Message Offset
- Sample Position Offset
- Centroid Position Offset

DWord	Bit	Description
M0.7:0		Ignored

SIMD8 Per Slot Offset Message Payload

This message payload applies only to the SIMD8 Per Slot Offset message type. The message length is 2.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**
M1.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M1.6	31:0	**Slot 6 Y Pixel Offset**

DWord	Bit	Description
M1.5	31:0	**Slot 5 Y Pixel Offset**
M1.4	31:0	**Slot 4 Y Pixel Offset**
M1.3	31:0	**Slot 3 Y Pixel Offset**
M1.2	31:0	**Slot 2 Y Pixel Offset**
M1.1	31:0	**Slot 1 Y Pixel Offset**
M1.0	31:0	**Slot 0 Y Pixel Offset**

SIMD16 Per Slot Offset Message Payload

This message payload applies only to the SIMD16 Per Slot Offset message type. The message length is 4.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**
M1.7	31:0	**Slot 15 X Pixel Offset**
M1.6	31:0	**Slot 14 X Pixel Offset**

DWord	Bit	Description
M1.5	31:0	**Slot 13 X Pixel Offset**
M1.4	31:0	**Slot 12 X Pixel Offset**
M1.3	31:0	**Slot 11 X Pixel Offset**
M1.2	31:0	**Slot 10 X Pixel Offset**
M1.1	31:0	**Slot 9 X Pixel Offset**
M1.0	31:0	**Slot 8 X Pixel Offset**
M2.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M2.6	31:0	**Slot 6 Y Pixel Offset**
M2.5	31:0	**Slot 5 Y Pixel Offset**
M2.4	31:0	**Slot 4 Y Pixel Offset**
M2.3	31:0	**Slot 3 Y Pixel Offset**
M2.2	31:0	**Slot 2 Y Pixel Offset**
M2.1	31:0	**Slot 1 Y Pixel Offset**
M2.0	31:0	**Slot 0 Y Pixel Offset**
M3.7	31:0	**Slot 15 Y Pixel Offset**
M3.6	31:0	**Slot 14 Y Pixel Offset**
M3.5	31:0	**Slot 13 Y Pixel Offset**
M3.4	31:0	**Slot 12 Y Pixel Offset**
M3.3	31:0	**Slot 11 Y Pixel Offset**

DWord	Bit	Description
M3.2	31:0	**Slot 10 Y Pixel Offset**
M3.1	31:0	**Slot 9 Y Pixel Offset**
M3.0	31:0	**Slot 8 Y Pixel Offset**

Writeback Message

SIMD8

The response length for all SIMD8 messages is 2. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**
W1.4	31:0	**Barycentric[2] for Slot 4**

DWord	Bit	Description
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0**

SIMD16

The response length for all SIMD16 messages is 4. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**
W1.4	31:0	**Barycentric[2] for Slot 4**

DWord	Bit	Description
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0** Format = IEEE_Float
W2.7	31:0	**Barycentric[1] for Slot 15**
W2.6	31:0	**Barycentric[1] for Slot 14**
W2.5	31:0	**Barycentric[1] for Slot 13**
W2.4	31:0	**Barycentric[1] for Slot 12**
W2.3	31:0	**Barycentric[1] for Slot 11**
W2.2	31:0	**Barycentric[1] for Slot 10**
W2.1	31:0	**Barycentric[1] for Slot 9**
W2.0	31:0	**Barycentric[1] for Slot 8**
W3.7	31:0	**Barycentric[2] for Slot 15**
W3.6	31:0	**Barycentric[2] for Slot 14**
W3.5	31:0	**Barycentric[2] for Slot 13**
W3.4	31:0	**Barycentric[2] for Slot 12**
W3.3	31:0	**Barycentric[2] for Slot 11**
W3.2	31:0	**Barycentric[2] for Slot 10**
W3.1	31:0	**Barycentric[2] for Slot 9**
W3.0	31:0	**Barycentric[2] for Slot 8**

Shared Functions - Unified Return Buffer (URB)

The Unified Return Buffer (URB) is a general-purpose buffer used for sending data between different threads, and, in some cases, between threads and fixed-function units (or vice-versa). A thread accesses the URB by sending messages.

URB Size

An URB entry is a logical entity within the URB, referenced by an entry handle and comprised of some number of consecutive rows. A row corresponds in size to a 256-bit EU GRF register. Read/write access to the URB is generally supported on a row-granular basis.

URB Size	URB Rows	URB Rows when SLM Enabled
See the Configurations volume.		

URB Access

The URB can be written by the following agents:

- Command Stream (CS) can write constant data into Constant URB Entries (CURBEs) as a result of processing CONSTANT_BUFFER commands.
- The Video Front End (VFE) fixed-function unit of the Media pipeline can write thread payload data in to its URB entries.
- The Vertex Fetch (VF) fixed-function unit of the 3D pipeline can write vertex data into its URB entries.
- GEN4 threads can write data into URB entries via URB_WRITE messages sent to the URB shared function.

The URB can be read by the following agents:

- The Thread Dispatcher (TD) is the main source of URB reads. As a part of spawning a thread, pipeline fixed-functions provide the TD with a number of URB handles, read offsets, and lengths. The TD reads the specified data from the URB and provide that data in the thread payload pre-loaded into GRF registers.
- The Geometry Shader (GS) and Clipper (CLIP) fixed-function units of the 3D pipeline can read selected parts of URB entries to extract vertex data required by the pipeline.
- The Windower (WM) FF unit reads back depth coefficients from URB entries written by the Strip/Fan unit.

Programming Note	
Context:	URB Access
The CPU cannot read the URB directly.	

State

The URB function is stateless, with all information required to perform a function being passed in the write message.

See URB Allocation (*Graphics Processing Engine*) for a discussion of how the URB is divided amongst the various fixed functions.

URB Messages

This section documents the global aspects of the URB messages. The actual data stored in URB entries differs for each fixed function – refer to *3D Pipeline* and the fixed-function chapters or details on 3D URB data formats and *Media* for media-specific URB data formats.

URB Handles: Unlike prior products where the URB handle contents was not specified for software use, URB handles are now specified as offsets into the URB partition in the L3 cache, in 512-bit units. Thus, kernels can now perform math operations on URB handles.

The **End of Thread** bit in the message descriptor may be set on URB messages only in threads dispatched by the vertex shader (VS), hull shader (HS), domain shader (DS), and geometry shader (GS). The **End of Thread** bit cannot be set on URB_READ* or URB_ATOMIC* messages.

Execution Mask. The low 8 bits of the execution mask on the send instruction determines which DWords from each write data phase are written or which DWords from each read phase are written to the destination GRF register. The execution mask is ignored on URB_ATOMIC* messages, because this is a scalar operation that is always enabled.

Out-of-Bounds Accesses. Reads to addresses outside of the URB region allocated in the L3 cache return 0. Writes to addresses outside of the URB region are dropped and do not modify any URB data.

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
URB Read HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Write HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
URB Read OWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Write OWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Atomic MOV	yes	N/A	N/A	handle + URBoffset	1
URB Atomic INC	yes	N/A	N/A	handle + URBoffset	1
URB Atomic ADD	yes	N/A	N/A	handle + URBoffset	1

"offset" is in the message payload, and is per-slot.

"handle" is the handle address in the message header.

"URBoffset" is the **Global Offset** field in the URB message descriptor.

Execution Mask

The Execution Mask specified in the 'send' instruction determines which DWords within each message register are read/written to the URB.

Message Descriptor

Bit	Description
19	**Header Present** This bit must be 1 for all URB messages.
18	Ignored.
17	**Per Slot offset:** If clear, the slot offset fields in the header are ignored. If set the slot offset fields are added to the global offset to obtain the overall offset. **Programming Notes:** • This bit must be 0 for URB_ATOMIC_* messages.

Bit	Description
16	Ignored.
15	**Swizzle Control.** This field specifies which “swizzle” operation is performed on the write data. It indirectly specifies whether one or two handles are valid. 0: URB_NOSWIZZLE The message accesses a single URB entry (using **URB Handle 0**). 1: URB_INTERLEAVED The message accesses two URB entries. The data is interleaved such that the upper DWords (7:4) of each 256-bit unit contain data associated with **URB Handle 1**, and the lower DWords (3:0) contain data associated with **URB Handle 0**. **Programming Notes:** • This bit must be 0 for URB_ATOMIC_* messages.
14:4	**Global Offset.** This field specifies a destination offset (in 256-bit units) from the start of the URB entry(s), as referenced by **URB Handle** *n*, at which the data (if any) is written to or read from. When URB_INTERLEAVED is used, this field provides a 256-bit granular offset applied to both URB entries. If the **Per Slot Offset** bit is set, this offset is added to the per-slot offsets in the header to obtain the overall offset. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. For the URB_ATOMIC* messages, this offset is in 32-bit units instead of 256-bit units. Format = U11 Range = [0, 1023] for URB_*_HWORD messages. Range = [0, 2047] for URB_*_OWORD messages. Range = [0, 2047] for URB_ATOMIC* messages.
3:0	**URB Opcode** 0: URB_WRITE_HWORD 1: URB_WRITE_OWORD 2: URB_READ_HWORD 3: URB_READ_OWORD 4: URB_ATOMIC_MOV 5: URB_ATOMIC_INC 6: URB_ATOMIC_ADD 7: URB_SIMD8_WRITE (see URB_SIMD8_* for descriptor details)

Bit	Description
	8: URB_SIMD8_READ (see URB_SIMD8_* for descriptor details) 9-15: Reserved

URB_WRITE and URB_READ

The URB_WRITE* and URB_READ* messages share the same header definition. URB_WRITE has additional payload containing the write data, but has no writeback message. URB_READ has no payload beyond the header (message length is always one), but always has a writeback message. URB_WRITE_SIMD4x2 has a single-phase payload with the per-slot offsets followed by the write data, and has no writeback message. URB_READ_SIMD4x2 has a single phase payload containing the per-slot offsets.

Message Header

M0.5[7:0] bits in message header are used for enabling DWs in cull test, at HDC unit by HS kernel, while writing TF data using URB write messages. Cull test is performed on outside TF and HS kernel set the appropriate DW enable, which carry the TF for different domain types. When DW is enabled and if cull test is positive, HS stage will be informed by HDC unit, to cull the HS handle early at HS stage itself.

DWord	Bits	Description
M0.5	31:17	Ignored
	16	**High OWORD Enable** For URB_READ_OWORD and URB_WRITE_OWORD with NOSWIZZLE indicates whether the 128 bits of the GRF register is used. 0: 1 OWord, read into or written from the low 128 bits of the GRF register. 1: 1 OWord, read into or written from the high 128 bits of the GRF register.
	15	Vertex 1 DATA [3] / Vertex 0 DATA[7] Channel Mask When **Swizzle Control** = URB_INTERLEAVED this bit controls Vertex 1 DATA[3]. When **Swizzle Control** = URB_NOSWIZZLE this bit controls Vertex 0 DATA[7]. This bit is ANDed with the corresponding channel enable to determine the final channel enable. For the URB_READ_OWORD & URB_READ_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be included in the writeback message. For the URB_WRITE_OWORD & URB_WRITE_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be written to the surface. 0: Vertex 1 DATA [3] / Vertex 0 DATA[7] channel not included. 1: Vertex DATA [3] / Vertex 0 DATA[7] channel included.
	14	**Vertex 1 DATA [2] Channel Mask**
	13	**Vertex 1 DATA [1] Channel Mask**

DWord	Bits	Description
	12	**Vertex 1 DATA [0] Channel Mask**
	11	**Vertex 0 DATA [3] Channel Mask**
	10	**Vertex 0 DATA [2] Channel Mask**
	9	**Vertex 0 DATA [1] Channel Mask**
	8	**Vertex 0 DATA [0] Channel Mask**
	7:0	**Reserved**
M0.4	31:0	**Slot 1 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 1**, at which the data will be accessed. This field is ignored unless **Swizzle Control** is set to URB_INTERLEAVED. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
M0.3	31:0	**Slot 0 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 0**, at which the data will be accessed. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
M0.1	31:16	Reserved.
	15:14	Reserved.
	13:0	**URB Handle 1.** This is the URB handle where channel 1's results are to be written or read. This field is ignored unless **Swizzle Control** indicates interleave mode.
M0.0	31:16	Reserved.
	15:14	Reserved.
	13:0	**URB Handle 0.** This is the URB handle where channel 0's results are to be written or read.

URB_WRITE_HWORD Write Data Payload

Programming Restriction: The write data payload can be between 1 and 8 message phases long.

For the URB_WRITE_HWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into consecutive URB locations (no data swizzling applied).

Programming Note	
Context:	URB_NOSWIZZLE
The URB function will use (not ignore) the Channel Enables associated with this message.	

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
M1.7	31:0	**Vertex Data [7]**
M1.6	31:0	**Vertex Data [6]**
M1.5	31:0	**Vertex Data [5]**
M1.4	31:0	**Vertex Data [4]**
M1.3	31:0	**Vertex Data [3]**
M1.2	31:0	**Vertex Data [2]**
M1.1	31:0	**Vertex Data [1]**
M1.0	31:0	**Vertex Data [0]**
M2.7	31:0	**Vertex Data [15]**
M2.6	31:0	**Vertex Data [14]**

DWord	Bit	Description
M2.5	31:0	**Vertex Data [13]**
M2.4	31:0	**Vertex Data [12]**
M2.3	31:0	**Vertex Data [11]**
M2.2	31:0	**Vertex Data [10]**
M2.1	31:0	**Vertex Data [9]**
M2.0	31:0	**Vertex Data [8]**
…		**…**
Mn.7	31:0	**Vertex Data [8(n-1)+7]**
Mn.6	31:0	**Vertex Data [8(n-1)+6]**
Mn.5	31:0	**Vertex Data [8(n-1)+5]**
Mn.4	31:0	**Vertex Data [8(n-1)+4]**
Mn.3	31:0	**Vertex Data [8(n-1)+3]**
Mn.2	31:0	**Vertex Data [8(n-1)+2]**
Mn.1	31:0	**Vertex Data [8(n-1)+1]**
Mn.0	31:0	**Vertex Data [8(n-1)+0]**

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

Programming Note	
Context:	URB_INTERLEAVED
• The URB function will use (not ignore) the Channel Enables associated with this message. • Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.	

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**
M2.7	31:0	**Vertex 1 Data [7]**
M2.6	31:0	**Vertex 1 Data [6]**
M2.5	31:0	**Vertex 1 Data [5]**
M2.4	31:0	**Vertex 1 Data [4]**
M2.3	31:0	**Vertex 0 Data [7]**
M2.2	31:0	**Vertex 0 Data [6]**
M2.1	31:0	**Vertex 0 Data [5]**
M2.0	31:0	**Vertex 0 Data [4]**
...		**...**
Mn.7	31:0	**Vertex 1 Data [4(n-1)+3]**
Mn.6	31:0	**Vertex 1 Data [4(n-1)+2]**
Mn.5	31:0	**Vertex 1 Data [4(n-1)+1]**

DWord	Bit	Description
Mn.4	31:0	**Vertex 1 Data [4(n-1)+0]**
Mn.3	31:0	**Vertex 0 Data [4(n-1)+3]**
Mn.2	31:0	**Vertex 0 Data [4(n-1)+2]**
Mn.1	31:0	**Vertex 0 Data [4(n-1)+1]**
Mn.0	31:0	**Vertex 0 Data [4(n-1)+0]**

URB_READ_HWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

Programming Restriction: The writeback message can be between 1 and 8 message phases long.

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
W0.7	31:0	**Vertex Data [7]**
W0.6	31:0	**Vertex Data [6]**

DWord	Bit	Description
W0.5	31:0	**Vertex Data [5]**
W0.4	31:0	**Vertex Data [4]**
W0.3	31:0	**Vertex Data [3]**
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**
W0.0	31:0	**Vertex Data [0]**
W1.7	31:0	**Vertex Data [15]**
W1.6	31:0	**Vertex Data [14]**
W1.5	31:0	**Vertex Data [13]**
W1.4	31:0	**Vertex Data [12]**
W1.3	31:0	**Vertex Data [11]**
W1.2	31:0	**Vertex Data [10]**
W1.1	31:0	**Vertex Data [9]**
W1.0	31:0	**Vertex Data [8]**
...		**...**
Wn.7	31:0	**Vertex Data [8n+7]**
Wn.6	31:0	**Vertex Data [8n+6]**
Wn.5	31:0	**Vertex Data [8n+5]**
Wn.4	31:0	**Vertex Data [8n+4]**
Wn.3	31:0	**Vertex Data [8n+3]**
Wn.2	31:0	**Vertex Data [8n+2]**

DWord	Bit	Description
Wn.1	31:0	**Vertex Data [8n+1]**
Wn.0	31:0	**Vertex Data [8n+0]**

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**
W0.3	31:0	**Vertex 0 Data [3]**
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**
W1.7	31:0	**Vertex 1 Data [7]**
W1.6	31:0	**Vertex 1 Data [6]**
W1.5	31:0	**Vertex 1 Data [5]**
W1.4	31:0	**Vertex 1 Data [4]**
W1.3	31:0	**Vertex 0 Data [7]**
W1.2	31:0	**Vertex 0 Data [6]**
W1.1	31:0	**Vertex 0 Data [5]**
W1.0	31:0	**Vertex 0 Data [4]**

DWord	Bit	Description
…		…
Wn.7	31:0	**Vertex 1 Data [4n+3]**
Wn.6	31:0	**Vertex 1 Data [4n+2]**
Wn.5	31:0	**Vertex 1 Data [4n+1]**
Wn.4	31:0	**Vertex 1 Data [4n+0]**
Wn.3	31:0	**Vertex 0 Data [4n+3]**
Wn.2	31:0	**Vertex 0 Data [4n+2]**
Wn.1	31:0	**Vertex 0 Data [4n+1]**
Wn.0	31:0	**Vertex 0 Data [4n+0]**

URB_WRITE_OWORD Write Data Payload

For the URB_WRITE_OWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into a single 128-bit URB location (no data swizzling applied).

Programming Note	
Context:	URB_NOSWIZZLE
The URB function will use (not ignore) the Channel Enables associated with this message.	

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
M1.7	31:0	Ignored
M1.6	31:0	Ignored
M1.5	31:0	Ignored
M1.4	31:0	Ignored
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
M1.7	31:0	**Vertex 0 Data [3]**
M1.6	31:0	**Vertex 0 Data [2]**
M1.5	31:0	**Vertex 0 Data [1]**
M1.4	31:0	**Vertex 0 Data [0]**
M1.3	31:0	Ignored
M1.2	31:0	Ignored
M1.1	31:0	Ignored
M1.0	31:0	Ignored

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

Programming Note	
Context:	URB_INTERLEAVED
• The URB function will use (not ignore) the Channel Enables associated with this message. • Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.	

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

URB_READ_OWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

Programming Note	
Context:	URB_READ_OWORD Writeback Message
Response Length must be set to 1.	

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
W0.7	31:0	Reserved (not written to GRF)
W0.6	31:0	Reserved (not written to GRF)
W0.5	31:0	Reserved (not written to GRF)
W0.4	31:0	Reserved (not written to GRF)
W0.3	31:0	**Vertex Data [3]**
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**
W0.0	31:0	**Vertex Data [0]**

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
W0.7	31:0	**Vertex Data [3]**
W0.6	31:0	**Vertex Data [2]**

DWord	Bit	Description
W0.5	31:0	**Vertex Data [1]**
W0.4	31:0	**Vertex Data [0]**
W0.3	31:0	Reserved (not written to GRF)
W0.2	31:0	Reserved (not written to GRF)
W0.1	31:0	Reserved (not written to GRF)
W0.0	31:0	Reserved (not written to GRF)

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**
W0.3	31:0	**Vertex 0 Data [3]**
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**

URB_ATOMIC

The URB_ATOMIC messages implement atomic operations on a single DWord in the URB. The location of the DWord within the URB is specified by the single URB handle and the **Global Offset** field in the message descriptor, which for these messages is a DWord offset from the URB handle. The DWord selected is operated on according to the following table:

URB Opcode	new_dst	ret
URB_ATOMIC_MOV	0	none
URB_ATOMIC_INC	old_dst + 1	old_dst

URB Opcode	new_dst	ret
URB_ATOMIC_ADD	old_dst + src0	old_dst

The previous contents of the DWord are returned in the destination register for operations that update the DWord value, such as URB_ATOMIC_INC. The URB_ATOMIC_MOV opcode does not return data (response length must be zero).

The URB_ATOMIC* messages consist only of the header. A single URB handle is specified.

Message Header

DWord	Bits	Description
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	**Source0 Data** Specifies the source 0 data for the atomic operation. This field is ignored for the URB_ATOMIC_INC message. Format = U32
M0.1	31:0	Ignored
M0.0	31:16	Ignored
	15:0	**URB Handle.** The URB handle to access.

Writeback Message

A writeback message is only returned for URB atomic operations that update the DWord value, such as URB_ATOMIC_INC. Only the low 32 bits of the destination GRF register are overwritten with the return data.

DWord	Bits	Description
W0.7:1		Reserved (not written to GRF)
W0.0	31:0	**Return Data** Specifies the value of the return data for the atomic operation. Format = U32

URB_SIMD8_Write and URB_SIMD8_Read

Programming Note	
Context:	URB_SIMD8_Write and URB_SIMD8_Read
The constant, sampler, and render caches are always non-coherent.	

Message Descriptor

Bit	Description
19	**Header Present** This bit must be set to **one** for all URB messages.
18	Ignored
17	**Per Slot offset Present:** If clear, then slot offset message phase is absent. If set then slot offset message phase is present and the per slot offsets are added to the global offset to obtain the overall offset.
16	**Ignored**
15	**Channel Mask Present:** If clear then the channel Mask Message phase is not present. If set then the channel mask message phase is present and will be used to mask data elements read or written.
14:4	**Global Offset.** This field specifies a destination offset (in 128-bit units) from the start of the URB entry(s), as referenced by **URB Handle** *n*, at which the data (if any) will be written to or read from. If the **Per Slot Offset** bit is set, this offset is added to the per-slot offsets in the header to obtain the overall offset. Format = U11 Range = [0, 2047]
3:0	**URB Opcode** 0: URB_WRITE_HWORD 1: URB_WRITE_OWORD 2: URB_READ_HWORD 3: URB_READ_OWORD 4: URB_ATOMIC_MOV 5: URB_ATOMIC_INC

Bit	Description
	6: URB_ATOMIC_ADD 7: URB_SIMD8_WRITE 8: URB_SIMD8_READ 9-15: Reserved

Message Header

DWord	Bits	Description
M0.7	31:16	Reserved
M0.7	15:14	Reserved
M0.7	13:0	**URB Handle 7.** This is the URB handle where channel 7's results are written or read.
M0.6	31:16	Reserved
M0.6	15:14	Reserved
M0.6	13:0	**URB Handle 6.** This is the URB handle where channel 6's results are written or read.
M0.5	31:16	Reserved
M0.5	15:14	Reserved
M0.5	13:0	**URB Handle 5.** This is the URB handle where channel 5's results are written or read.
M0.4	31:16	Reserved
M0.4	15:14	Reserved
M0.4	13:0	**URB Handle 4.** This is the URB handle where channel 4's results are written or read.
M0.3	31:16	Reserved
M0.3	15:14	Reserved
M0.3	13:0	**URB Handle 3.** This is the URB handle where channel 3's results are written or read.
M0.2	31:16	Reserved
M0.2	15:14	Reserved
M0.2	13:0	**URB Handle 2.** This is the URB handle where channel 2's results are written or read.
M0.1	31:16	Reserved
M0.1	15:14	Reserved
M0.1	13:0	**URB Handle 1.** This is the URB handle where channel 1's results are written or read.
M0.0	31:16	Reserved
M0.0	15:14	Reserved
M0.0	13:0	**URB Handle 0.** This is the URB handle where channel 0's results are written or read.

Per Slot Offset Message Phase

When the **Per Slot offset Present** bit in the descriptor is set then the Per slot offset message phase is sent by the EUs. The per slot message phase occurs immediately after the header.

DWord	Bit	Description
M1.7	31:0	**Slot 7 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 7**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.6	31:0	**Slot 6 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 6**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.5	31:0	**Slot 5 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 5**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.4	31:0	**Slot 4 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 4**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.3	31:0	**Slot 3 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 3**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.2	31:0	**Slot 2 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 2**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.1	31:0	**Slot 1 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 1**, at which

DWord	Bit	Description
		the data will be accessed. Format = U32 Range = [0, 2047]
M1.0	31:0	**Slot 0 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 0**, at which the data will be accessed. Format = U32 Range = [0, 2047]

Channel Mask Message Phase

When the **Channel Mask Present** bit in the descriptor is set then the channel mask message phase is sent by the EUs. The channel mask message phase occurs after the per slot message phase if the per slot message phase exists else it occurs after the header.

DWord	Bit	Description
M2.7	31:24	**Reserved.**
	23	Vertex 7 DATA [7] Channel Mask This bit is ANDed with the corresponding channel enable to determine the final channel enable. For the URB_SIMD8_READ messages, when final channel enable is 1 it indicates that Vertex 7 DATA [7] will be included in the writeback message. For the URB_SIMD8_WRITE messages, when final channel enable is 1 it indicates that Vertex 7 DATA [7] will be written to the surface. 0: Vertex 7 DATA [7] channel not included 1: Vertex 7 DATA [7] channel included
	22	**Vertex 7 DATA [6] Channel Mask**
	21	**Vertex 7 DATA [5] Channel Mask**
	20	**Vertex 7 DATA [4] Channel Mask**
	19	**Vertex 7 DATA [3] Channel Mask**
	18	**Vertex 7 DATA [2] Channel Mask**
	17	**Vertex 7 DATA [1] Channel Mask**

DWord	Bit	Description
	16	**Vertex 7 DATA [0] Channel Mask**
	15:0	**Reserved.**
M2.6	31:24	**Reserved.**
	23:16	**Vertex 6 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.5	31:24	**Reserved.**
	23:16	**Vertex 5 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.4	31:24	**Reserved.**
	23:16	**Vertex 4 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.3	31:24	**Reserved.**
	23:16	**Vertex 3 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.2	31:24	**Reserved.**
	23:16	**Vertex 2 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.1	31:24	**Reserved.**
	23:16	**Vertex 1 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.0	31:24	**Reserved.**

DWord	Bit	Description
	23:16	**Vertex 0 DATA [7:0] Channel Mask**
	15:0	**Reserved.**

Workarounds

Issue
URB SIMD8 Channel Mask- URB SIMD8 Messages not supporting combinations of mixed EU settings
URB SIMD8 Channel Mask-Address corruption - URB SIMD8 Channel Mask-Address corruption

Write Data Payload

The write data payload can be between 1 and 8 message phases long.

DWord	Bit	Description
M3.7	31:0	**Vertex 7 DATA [0]**
M3.6	31:0	**Vertex 6 DATA [0]**
M3.5	31:0	**Vertex 5 DATA [0]**
M3.4	31:0	**Vertex 4 DATA [0]**
M3.3	31:0	**Vertex 3 DATA [0]**
M3.2	31:0	**Vertex 2 DATA [0]**
M3.1	31:0	**Vertex 1 DATA [0]**
M3.0	31:0	**Vertex 0 DATA [0]**
...	...	**...**
M10.7	31:0	**Vertex 7 DATA [7]**
M10.6	31:0	**Vertex 6 DATA [7]**
M10.5	31:0	**Vertex 5 DATA [7]**
M10.4	31:0	**Vertex 4 DATA [7]**

DWord	Bit	Description
M10.3	31:0	**Vertex 3 DATA [7]**
M10.2	31:0	**Vertex 2 DATA [7]**
M10.1	31:0	**Vertex 1 DATA [7]**
M10.0	31:0	**Vertex 0 DATA [7]**

Writeback Message

The writeback message can be between 1 and 8 message phases long.

DWord	Bit	Description
M0.7	31:0	**Vertex 7 DATA [0]**
M0.6	31:0	**Vertex 6 DATA [0]**
M0.5	31:0	**Vertex 5 DATA [0]**
M0.4	31:0	**Vertex 4 DATA [0]**
M0.3	31:0	**Vertex 3 DATA [0]**
M0.2	31:0	**Vertex 2 DATA [0]**
M0.1	31:0	**Vertex 1 DATA [0]**
M0.0	31:0	**Vertex 0 DATA [0]**
...	...	**...**
M7.7	31:0	**Vertex 7 DATA [7]**
M7.6	31:0	**Vertex 6 DATA [7]**
M7.5	31:0	**Vertex 5 DATA [7]**
M7.4	31:0	**Vertex 4 DATA [7]**
M7.3	31:0	**Vertex 3 DATA [7]**

DWord	Bit	Description
M7.2	31:0	**Vertex 2 DATA [7]**
M7.1	31:0	**Vertex 1 DATA [7]**
M7.0	31:0	**Vertex 0 DATA [7]**

Message Gateway

The Message Gateway has these functions:

- Barrier messages for thread-to-thread synchronization
- Timestamp provide a counter with a fixed frequency not dependent on the core clock.
- End of Thread messages (sent to Thread Spawner), to exit the GPGPU and Media threads.

Message Descriptor

The following message descriptor applies to all messages supported by Message Gateway.

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>19</td><td>Header Present. This bit must be 0 for all Message Gateway messages.</td></tr>
<tr><td>18:17</td><td>Ignored.</td></tr>
<tr><td>16:3</td><td>Reserved: MBZ</td></tr>
<tr><td>2:0</td><td>SubFuncID. Identify the supported sub-functions by Message Gateway. Encodings are:
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>011b</td><td>GetTimeStamp</td><td>Read absolute and relative timestamps.</td></tr>
<tr><td>100b</td><td>BarrierMsg</td><td>Record an additional thread reaching the barrier</td></tr>
</table>
All other encodings Reserved.</td></tr>
</table>

GetTimeStamp Message

The GetTimeStamp message gives the ability for a requester thread to read the timestamps back from the message gateway. The message consists of a single 256-bit message payload.

AbsoluteTimeLap is based on an absolute wall clock in unit of nSec/uSec that is independent of context switch or GPU frequency adjustment. Message Gateway shares the same GPU timestamp. Details can be found in the TIMESTAMP register section in *vol1c Memory Interface and Command Stream*.

RelativeTimeLap is based on a relative time count that is counting the GPU clocks for the context. The relative time count is saved/restored during context switch.

Message Payload

DWord	Bits	Description
M0.5	31	**Return to High GRF**: 0: The return 128-bit data goes to the first half of the destination GRF register. 1: The return 128-bit data goes to the second half of the destination GRF register.
	30:8	Reserved: MBZ
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway. This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Writeback Message to Requester Thread

As the writeback message is only sent if the **AckReq** bit in the message descriptor is set, **AckReq** bit must be set for this message.

Only half of the destination GRF register is updated (via write-enables). The other half of the register is not changed. This is determined by the **Return to High GRF** control field.

Writeback Message if Return to High GRF is set to 0:

DWord	Bit	Description
W0.7:4		Reserved (not overwritten)
W0.3	31:0	**RelativeTimeLapHigh:** This field returns the MSBs of time lap for the relative clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12

DWord	Bit	Description
W0.2	31:20	**RelativeTimeLapLow:** This field returns the LSBs of time lap for the relative clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ
W0.1	31:0	**AbsoluteTimeLapHigh:** This field returns the MSBs of time lap for the absolute clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.0	31:20	**AbsoluteTimeLapLow:** This field returns the LSBs of time lap for the absolute clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ

Writeback Message if Return to High GRF is set to 1:

DWord	Bit	Description
W0.7	31:0	**RelativeTimeLapHigh**
W0.6	31:20	**RelativeTimeLapLow**
	19:0	Reserved : MBZ
W0.5	31:0	**AbsoluteTimeLapHigh**
W0.4	31:20	**AbsoluteTimeLapLow**
	19:0	Reserved : MBZ
W0.3:0		Reserved : MBZ

BarrierMsg Message

The BarrierMsg message gives the ability for multiple threads to synchronize their progress. This is useful when there are data shared between threads. The message consists of a single 256-bit message payload.

Upon receiving one such message, Message Gateway increments the Barrier counter and marks the Barrier requester thread. There is no immediate response from the Message Gateway when the incremented counter is not equal to the terminating thread count. When the incremented counter value does equal the **Barrier Thread Count**, Message Gateway sends a response back to all the Barrier requester threads and resets the Barrier count to zero.

Programming Note	
Context:	BarrierMsg Message
The Message Gateway assumes that the barrier ID sent in barrier message payload is valid and was allocated by TSG. In the event of a programming error specifying an invalid barrier ID, the results are undefined, and may cause the Message Gateway to stop responding to barrier messages for any thread that it services.	

Message Payload

DWord	Bits	Description
M0.5	31:0	**Ignored**
M0.4	31:0	**Ignored**
M0.3	31:0	**Predicate Mask.** This field has a bit set per SIMD channel that passes the predicate. For SIMD8 and SIMD16 the rest of the bits must be 0. This field is ignored for non-predicated barriers.
M0.2	31	**Barrier ID MSB.** This field is bit[4] of the BarrierID, for full 5-bit barrier ID it should be combined with Barrier ID[3:0]. Format: U1
	30	**Predicate Mask Enable.** This bit indicates that the barrier is a predicated barrier and the SIMD channels passing the predicate should be summed. All threads sending this message to the same barrier should have an identical value for this field, and must specify a response length of 1 for the predicate sum response. Note that Global Barriers must not have the Predicate Mask Enable bit set.
	27:24	**BarrierID.** This field indicates which one from the 16 Barrier States is updated. There is a fifth bit for the BarrierID, in bit 31. Format: U4 Note: This field location matches with that of R0 header.
	23:16	**Ignored**
	15	**Barrier Count Enable.** Allows the message to reprogram the terminating barrier count. If set, the stored value of the terminating barrier count is set to the value of Barrier Count field (below), and used for this barrier operation. If clear, the stored value of the terminating barrier count is not modified and the stored value is used for this barrier operation. **Programming Note:** This control is intended only for Hull Shader threads. Do not use this control if the barrier is linked with other barriers in other subslices. Format: Enable
	14:9	**Barrier Count.** If Barrier Count Enable is set, this field specifies the terminating barrier count. Otherwise this field is ignored. All threads that belong to a single barrier must deliver the same value for this field for a particular barrier iteration.

DWord	Bits	Description
	8:0	**Ignored**
M0.1	31:0	**Ignored**
M0.0	31:4	**Ignored**

Writeback Message to Requester Thread

The writeback message is sent only if the **AckReq** bit in the message descriptor is set.

Programming Note	
Context:	Writeback Message to Requester Thread
AckReq must not be used with predicate barriers, because the predicate barrier response is sent to the GRF writeback location instead of the Ack.	

DWord	Bits	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID.** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal). Other encodings are reserved.

Broadcast Writeback Message

Description
When the count for a Barrier reaches Barrier.Count, the Message Gateway sends the notification bit N0 to each EU/Thread that reached the barrier. A Barrier Return Byte is not sent.
This response message is sent back only if the Gateway Barrier Message specifies that this is a predicated barrier. This response is written to the GRF writeback location, and the response length specified in the send message to the EU must be 1.

DWord	Bits	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:16	Reserved (not overwritten)
	15:0	**Predicated Barrier Mask Sum.** This field is a sum of the predicate mask bits sent by each thread. This field (and the DW containing it) is not written if the barrier is not marked as a predicated barrier. The kernel should compare this field to 0 for the predicated OR function and compare it to the workgroup size for the predicated AND function.

Media Sampler

This section describes the functionality of the Media Sampler.

Shared Functions – Video Motion Estimation

The Video Motion Estimation (VME) engine is a shared function that provides motion estimation services. It includes motion estimation for various block sizes and also standard specific operations such as

- Motion estimation and mode decision for AVC
- Intra prediction and mode decision for AVC
- Motion estimation and mode decision for MPEG2
- Motion estimation and mode decision for VC1

The motion estimation engine may also be used for other coding standards or other video processing applications.

Theory of Operation

VME performs a sequence of operations to find the best mode for a given macroblock. Each operation step can be enabled/disabled through the control of the income message. Early termination, skipping of subsequent operation steps, is also supported when certain search criteria are met.

VME contains the following operation steps:

1. Skip check
2. IME: Integer motion estimation
3. FME: Fractional motion estimation
4. BME: Bidirectional motion estimation
5. IPE: Intra prediction estimation (AVC only)

Shape Decision

As a terminology, we call sub-block shapes: 8x4, 4x8, and 4x4 *minor shapes* (corresponding to *sub-partitions* of 8x8 sub-macroblock), and 16x16, 16x8, 8x16, and 8x8 *major shapes* (corresponding to *sub-macroblocks* of a 16x16 macroblock)**.**

If the maximal allowed number of motion vectors **MaxNumMVs** (**MaxNumMVs** = **MaxNumMVsMinusOne** + 1) is less than 4, we will set minor MV flag off: **MinorMVsFlag** = 0, *i.e.* no minor motion vectors will be generated.

The reason of having this parameter **MaxNumMVs** is due to high level AVC conformance restrictions for certain profiles: *the total number of motion vectors of any two consecutive macroblocks not exceeding 16 (or 32)*. The mechanism here allows a reasonable degree of user control. In disable cases, **MaxNumMVs** should be set to 32.

In the coding process of VME, the shape decision is done in multiple locations:

1. After IME and before FME, intermediate shape decision is performed to reduce the FME searching candidates
2. After FME and before BME, existing shape decision is revised among the remaining candidates and to see if there is further reduction.
3. Final shape decision is done after BME.

Partition decision before BME uses unidirectional motion vector count to meet **MaxNumMVs** requirement. Adding BME for the partition candidates may exceed **MaxNumMVs**. As BME is performed on a block by block basis using the block order for a given partition, BME step for a given block is skipped and the best unidirectional motion vectors are used for the block if the overall motion vector count exceeds **MaxNumMVs** when that particular block is switched to bidirectional. The process continues to the last block of the partition.

Note: This is a sub-optimal solution to simplify the hardware implementation. For some cases, bidirectional modes with larger sub-partitions might be better than unidirectional modes with finer sub-partitions.

The VME implementation has the following restriction: Multiple partition candidates are only enabled if **PartCandidateEn** is set. And this only applies to source block of size 16x16.

If **PartCandidateEn** is not set, only the best partition is kept in state 1 (after IME) above and carried through FME and BME. In other words, FME if enabled only operates on one partition candidate, and BME if enabled only operates on one partition candidate. Bidirectional mode check only applies to the partition candidates that meet the bidirectional restriction provided by **BiSubMbPartMask**. For example, if a minor partition determined based on best unidirectional cost function is not 8x8 but one of 4x8, 8x4 or 4x4, VME skips the bidirectional mode check.

If **PartCandidateEn** is set, up to two sets of candidates are maintained by VME hardware, if the second best partition candidate is within **PartToleranceThrhd** from the best one. The second best partition is selected only from the two major partition candidates based on the unidirectional motion vector count, subject to that the major partition is enabled:

- 1MV: The 16x16 partition
- 4MV: The 4x(8x8) partition with no minor shape

The following partitions are not supported as alternative partition.

- 2MV: The best of 2x(16x8) and 2x(8x16) partitions
- More than 4MV: The best of all 4x(8x8) partitions with at least one 8x8 having minor shape of 8x4, 4x8 or 4x4

Major Shape Decision Prior to FME

Now considering the best of each 8x8 is done, and we have the total cost-adjusted-distortion for this sub-block level partition. Now among the four choices: the resulting 8x8 sub-partitioning, one 16x16, two 16x8, and two 8x16, the one gives the best cost-adjusted-distortion, will determine the final decision of partitioning shape. Any among these four, if its cost-adjusted-distortion is within *the intermediate*

tolerance (which is a predefined system state) from the best distortion will be marked as **candidate shape**s.

Notice that, when the intermediate tolerance is set to 0, only the best shape will be selected as the candidate. When the intermediate tolerance is large, all four shapes will become candidates.

Assume we have all the distortions for majors enumerated in DistoMajor[k], where k = 0, 1, 2, 3, 4, and 5, for 16x16, 16x8, 8x16, the combined minors, 16x8 field, and 8x8 field respectively. Assume BestDisto is equal to the minimal of the six values DistoMajor[k], for k = 0, ...5. Assume the intermediate tolerance is IntTol, the major shape k is a candidate shape if and only if DistoMajor[k]<=BestDisto+IntTol.

Shape Update after FME

Among all the candidate shapes, we recheck the distortion, if any of them is no longer with in the intermediate tolerance **DistortionTolerance** from the best choice; we drop it for reduced calculation.

Final Code Decision after BME

For any given candidate shape, for each motion vector, if we do have improved distortion by switch from the single direction to bi-direction, then we do it, unless the increased number of motion vectors hits above **MaxNumMVs**; in this case, we take as many as possible first the ones generate the most improvement.

Then, we choose the best among the improved candidate shapes.

Surfaces

The data elements accessed by VME are called “surfaces”. Surfaces are accessed using the surface state model.

VME uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine. A **Binding Table Index** (specified in the message descriptor) of less than 255 is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE. SURFACE_STATE contains the parameters defining the surface to be accessed, including its location, format, and size.

State

BINDING_TABLE_STATE

VME uses the binding table to retrieve surface state. Refer to *Sampling Engine* for the definition of this state.

SURFACE_STATE

VME uses the surface state for current and reference surfaces. Refer to *Sampling Engine* for the definition of this state.

VME_STATE

This state structure contains the state used by the VME engine for data processing. VME state contains the motion search path location tables and rate-distortion weight look-up-tables. As the two sets of tables are fairly large, they are accessed as two separate states via state indexing mechanism so that applications can inter-mix the use of the search path tables and RDLUT tables.

Even though VME engine has its unique shared function ID (see Target Function ID field in the SEND instruction), the VME state is delivered through the Sampler State Pointer. When the General Purpose Pipe is used, the **Sampler State Pointer** is programmed in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command and delivered directly to Sampler/VME by hardware. This posts one usage limitation. As the VME state is overloaded on top of the Sampler State Pointer, VME messages cannot be intermixed with other Sampler messages.

Each VME state may contain up to 8 VME_SEARCH_PATH_LUT_STATE. When multiple VME_SEARCH_PATH_LUT_STATE are used, they need to be stored in memory contiguously. Each VME_SEARCH_PATH_LUT_STATE contains 32 dwords in comparison of 4 dwords of a Sampler State. When enabling sampler state pre-fetch (programming the **Sampler Count** field in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command), one VME_SEARCH_PATH_LUT_STATE is equivalent to 8 Samplers. Hardware may support up to two VME_SEARCH_PATH_LUT_STATE to be pre-fetched (See See 3D_Media_GPGPU chapter, Media_GPGPU_Pipeline for more details).

VME_SEARCH_PATH_LUT_STATE

Up to eight VME_SEARCH_PATH_LUT_STATE allowed for a message to select. Each state contains one set of search path locations, and four sets of rate distortion cost function LUT for various modes and rate distortion cost function LUT for motion vectors (relative to 'cost center'). Motion vector cost function is provided as a piece-wise-linear curve with only the values of the power-of-2 positions provided.

DWord	Bit	Description
0:13		**Search Path**
0	31:24	Search Path Location [3] (X, Y) – Relative distance from location [2]
	23:16	Search Path Location [2] (X, Y) – Relative distance from location [1]
	15:8	Search Path Location [1] (X, Y) – Relative distance from location [0]
	7:4	Search Path location [0] (Y) – specifies relative Y distance of the **next walk** from the starting position in unit of Search Unit (SU) in U4 Format = U4, (e.g. 0x3 + 0xE = 0x1)

DWord	Bit	Description
	3:0	Search Path Distance [0] (X) – specifies relative X distance of the next walk from the starting position in unit of SU. Format = U4
1:13		Search Path Location [4 – 55] (X, Y)
14:31		**RD LUT SET 0-4**
14	31:24	LUT_MbMode [9] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [8] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [9] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
15	31:24	LUT_MbMode [9] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [8] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [9] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
16	31:24	LUT_MbMode [3] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [2] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)

DWord	Bit	Description
	15:8	LUT_MbMode [1] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [0] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
17	31:24	LUT_MbMode [7] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MbMode [6] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MbMode [5] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MbMode [4] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
18	31:24	LUT_MV [3] – For MV = 4 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [2] – For MV = 2 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MV [1] – For MV = 1 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MV [0] – For MV = 0 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
19	31:24	LUT_MV [7] – For MV = 64 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [6] – For MV = 32 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)

DWord	Bit	Description
	15:8	LUT_MV [5] – For MV = 16 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MV [4] – For MV = 8 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
20-23		**Finish RD LUT SET 1**
24-27		**Finish RD LUT SET 2**
28-31		**Finish RD LUT SET 3**

The assignment of LUT_MbMode entries is according to the MbTypeEx definition:

Index to LUT_MbMode	MbTypeEx	Description	AVC	VC1	MPEG2
0	MODE_INTRA_NONPRED	For INTRA8x8 and INTRA4x4 only. Added per 8x8 for INTRA8x8, and per 4x4 for INTRA4x4	Yes	n/a	n/a
1	MODE_INTRA MODE_INTRA_16x16	Added per 16x16 macroblock	Yes	Yes	Yes
2	MODE_INTRA_8x8	Added per 16x16 macroblock	Yes	n/a	n/a
3	MODE_INTRA_4x4	Added per 16x16 macroblock	Yes	n/a	n/a
8	MODE_INTER MODE_INTER_16x16	Added per 16x16 macroblock	Yes	Yes	Yes
9	MODE_INTER_BWD	Added for Refldx (per partition for major type or 8x8 for minor types)	Yes	Yes	Yes
4	MODE_INTER_16x8 MODE_INTER_8x16	Added per 16x16 macroblock	Yes	n/a	n/a
5	MODE_INTER_8x8q	Added per 8x8 subblock	Yes	Yes	n/a
6	MODE_INTER_8x4q	Added per 8x8 subblock	Yes	n/a	n/a
6	MODE_INTER_4x8q	Added per 8x8 subblock	Yes	n/a	n/a
7	MODE_INTER_4x4q	Added per 8x8 subblock	Yes	n/a	n/a
6	MODE_INTER_FIELD_16x8	Added per 16x16 macroblock	n/a	?	Yes
7	MODE_INTER_FIELD_8x8q	Added per 16x16 macroblock	n/a	n/a	n/a

The value of each byte of the LUTs will be viewed as a pair of 4-bit units: (shift, base), and constructed as base « shift.

For example, an entry 0x4A represents the value (0xA«0x4) = 10*16 = 160. Encoded value must fit in **12**-bits (unsigned number); otherwise, the hardware behavior is undefined.

The only exception is for Index of 9, MODE_INTER_BWD, which is used as a bias for the two search directions. It is a signed number instead, in the form of (SU3U4) = (sign, shift, base). The sign bit indicates whether the bias is added to the forward (if sign = 1) or the backward (if sign = 0). The bias has a magnitude of (base « shift), which has 11-bits precision. It should be noted that the number is always added, there is no subtraction.

Intra Modes only apply to AVC standard. The mode penalty doesn't apply to Skip Mode Checking. Note that while other mode penalty applies to a fixed macroblock partition, MODE_INTRA_NONPRED applies to all three intra modes. It is a constant cost adder for intra-mode coding regardless of the block size.

For source block that is less than 16x16 (like a 16x8 source block), the proper mode penalty that is stated as "added per 16x16 macroblock" is added once to the source block (like MODE_INTER_16x8 is added once to a 16x8 source block). It will not be divided by the source block size.

The LUT_MV is added to all motion vector coordinate deltas in quarter-pel unit except for the SKIP mode, which no costing penalty applies. Given motion vector coordinate, e.g. *mvx*, which is in quarter-pel precision (S5.2), the mv delta is defined to be its difference from the given costing center, e.g. *ccx*, and the costing penalty is applied to $dx = |mvx-ccx|$. The cost penalty is a piecewise linear interpolation from the LUT_MV table whereas the values on power-of-2 integer samples are provided. The piecewise linear interpolation is performed using quarter-pel precision, while the LUT_MV are only provided for the given power-of-2 integer positions. The maximum distance provided in the table is 64 pixels. A linear ramp with gradient of 1 on integer distance is applied for bigger distances with maximum penalty capped to 0x3FF (10 bits). Thus,

Costing_penalty_x = LUT_MV[int(dx)], if $dx < 3$ and dx = int(dx);

Costing_penalty_x = LUT_MV[p+1], else if $dx = 2^p$, for any p£6;.

Costing_penalty_x = LUT_MV[p+1] + ((LUT_MV[p+2] – LUT_MV[p+1])*k)»p,

else if $dx = 2^p+k$, for any $p<6$ and $k<.2^p$, and

Costing_penalty_x = min (LUT_MV[7] + int(dx)– 64, 255), else if $dx > 64$.

The total costing penalty for a motion vector is

Costing_penalty = Costing_penalty_x + Costing_penalty_y

As a convention, a (0,0) relative search path distance (meaning a repeat search path location) is treated as the ending of the search path. Or the search path may also end when **Max Predetermined Search Path Length** is reached, or one of the Early Success conditions is reached**.**

Software must program the search path to terminate with at least one (0,0).

Glossary of Messages

This section describes the glossary of messages in regard to Media Sampler.

Programming Note	
Context:	Glossary of Messages
Use of any messages to the Video Motion Estimation function while there are any messages to any sampler function is not allowed.	

Universal Input Message Phases

Major changes from the previous generation:

- Many fields are only required for one or two of the message types.
- MV cost and mode cost are moved into the message payload.
- RefID per block are new inputs.
- Enables for forward transform skip check, chroma searching.
- Thresholds and control data for forward transform skip check.
- Many of the performance thresholds have been removed (IME success, skip success, etc).

ValidMsgType = "..." identifies the given field is required for each message type. Hardware ignores these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

DWord	Bits	Description
M0.5	31:24	Reference Region Height (RefHeight): This field specifies the reference region height in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in vertical direction. The value must be a multiple of 4. ValidMsgType = IME Format = U8 Range = [20, 64]
	23:16	Reference Region Width (RefWidth): This field specifies the search region width in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in horizontal direction. The value must be a multiple of 4. ValidMsgType = IME Format = U8 Range = [20, 64] Note: Please make sure the reference windows are not completely outside of the video frame. In

DWord	Bits	Description
		that case, VME behavior is undefined. Note: Reference Window size must be <= Surface Size, otherwise VME behavior is undefined.
	15:8	Ignored
	7:0	Dispatch ID. This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. ValidMsgType = SIC, IME, FBR
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31	**2D 7 by 7 Enable** 0: Disable 1: Enable Notes: • Output phase W0 and W6 MBZ when this bit is asserted. • 2dsobel applies only to single reference and single start. • 2dsobel applies only to ref48x40.
	30:24	Sub-Macroblock Sub-Partition Mask (SubMbPartMask): This field defines the bit-mask for disabling sub-partition and sub-macroblock modes. The lower 4 bits are for the major partitions (sub-macroblock) and the higher 3 bits for minor partitions (with sub-partition for 4x(8x8) sub-macroblocks. xxxxxx1 : 16x16 sub-macroblock disabled xxxxx1x : 2x(16x8) sub-macroblock within 16x16 disabled xxxx1xx : 2x(8x16) sub-macroblock within 16x16 disabled xxx1xxx : 1x(8x8) sub-partition for 4x(8x8) within 16x16 disabled xx1xxxx : 2x(8x4) sub-partition for 4x(8x8) within 16x16 disabled x1xxxxx : 2x(4x8) sub-partition for 4x(8x8) within 16x16 disabled 1xxxxxx : 4x(4x4) sub-partition for 4x(8x8) within 16x16 disabled 1111111: Invalid Note: Invalid to have all partions disabled in the IME call. ValidMsgType = IME Usage Note: One example usage of only enabling 4x(4x4) sub-partition while all other partitions are disabled is for video processing, where parallel motion searches are performed for 16 4x4 blocks. For that no further block combination (into larger sub-partitions/sub-macroblocks) is needed.

DWord	Bits	Description
	23:22	Intra SAD Measure Adjustment (IntraSAD): This field specifies distortion measure adjustments used for the motion search SAD comparison. This field applies to both luma and chroma intra measurement. 00b: None 01b: Reserved 10b: Haar transform adjusted 11b: Reserved ValidMsgType = SIC
	21:20	Inter SAD Measure Adjustment (InterSAD):This field specifies distortion measure adjustments used for the motion search SAD comparison. This field applies to both luma and chroma intra measurement. 00b: None 01b: Reserved 10b: Haar transform adjusted 11b: Reserved ValidMsgType = SIC, IME, FBR, IDM Note: IDM msgs cannot have InterSAD set to 10b (Haar transform adjusted) if IdmSrcPixelMask is used. InterSAD must be set to 00b (None) if either IDMShapeMode5x5 or IDMShapeMode7x7 is enabled.
	19	Block-Based Skip Enabled: When this field is set on the skip thresholding passing criterion will be based on the maximal distortion of individual blocks (8x8's or 4x4's) instead of their sum (i.e. the distortion of 16x16). The block size is 8x8 if and only if the Transform 8x8 Flag is set to ON and the source size is 16x16. ValidMsgType = SIC
	18	BME disable for FBR Message (BMEDisableFBR): FBR messages that do not want bidirectional motion estimation performed will set this bit and VME will only perform fractional refinement on the shapes identified by subpredmode. Note: only the LSB of the subpredmode for each shape will be considred in FBR (a shape is either FWD or BWD as input of FBR, output however could change to BI if BME is enabled). 0 = BME enabled 1 = BME disabled ValidMsgType = FBR

DWord	Bits	Description
	17	Forward Transform Skip Check Enable (FTEnable): This field enables the forward transform calculation for skip check. It does not override the other skip calculations but it does decrease the performance marginially so don't enable it unless the transform is necessary. 0 = FT disabled 1 = FT enabled ValidMsgType = SIC
	16	**Process Inter Chroma Pixels Mode** (InterChroaZmaMode): This bit switches the inter operations from luma mode to chroma mode. All shapes sizes are referred to as UV pairs. For instance, the 4x4 shape is a 8x4 of pixel components (16 U and 16 V, interleaved vertically) and the 8x8 shape is a 16x8 of pixel components. MBMode is always 8x8. MBSubShape is either 8x8 or 4x4 indicated by LSB[1:0]. Bits[7:2] are MBZ. For MBSubShape of 4x4, SubPredMode is mapped to each 4x4 shape. Only 8x8 and 4x4 ModeCost are valid. Source block size is ignored. Streamin/streamout distortions are overloaded on 16x16 (Chroma8x8) and 8x8 (Chroma4x4). BilinearEnable is ignored (Chroma can only perform bilinear filtering) Restrictions when set: Intra operations are disabled (SIC), valid ref window sizes are 32x20, 24x24 (max Xsus), 16x32 (max Xsus), and 10x20 (max Xsus) (IME), adaptive is disabled (IME), no backward penalty cost (ALL), and only 4x4 and 8x8 shapes are valid (ALL). ValidMsgType = SIC IME, FBR
	15	Disable Field Cache Allocation: This field, when set to 1, disables the optimized field cache line method in the Sampler Cache for reference block data when RefAccess is 1 (field based). It is ignored by hardware if RefAccess is 0. 0 – Frame or field cache lines according to RefAccess 1 – Always frame cache lines ValidMsgType = IME, IDM
	14	**Skip Mode Type** For B_DIRECT_16x16, both motion vectors of the skip center pair 0 are used. For B_DIRECT_8x8s, all four skip center pairs are fully used (VME never tries to combine them with non-skip shapes from IME, FME, or BME).

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>0 : SKIP_1MVP – one MV pair for 16x16
1 : SKIP_4MVP – Four MV pairs for 8x8s (in this case and only this case, SkipCenter Delta 1-3 is used)
Note: SkipModeType should be programmed to 1MVP for non-16x16 Source size.
ValidMsgType = SIC</td></tr>
<tr><td></td><td>13:12</td><td>Sub-Pel Mode (SubPelMode)
This field defines the half/quarter pel modes. The mode is inclusive, ie., higher precision mode samples lower precision locations.
00b: Integer mode searching
01b: Half-pel mode searching
10b: Reserved
11b: Quarter-pel mode searching
ValidMsgType = FBR</td></tr>
<tr><td></td><td>11</td><td>Dual Search Path Option
Used only for dual record cases, this field flags whether two searching records uses the same or the different paths.
0: Use the same path as specified by the Search Path Location array
1: Use the different paths, the first one uses the even entries of the Search Path Location array and the second one uses the odd entries of the Search Path Location array.
ValidMsgType = IME</td></tr>
<tr><td></td><td>10:8</td><td>Search Control (SearchCtrl)
This field specifies how the motion search is performed.
ValidMsgType = IME
The following table shows the valid encodings. Other encodings are reserved.
<table>
<tr><th>Code</th><th>Mode</th></tr>
<tr><td>000b</td><td>Single reference, single record and single start.
Search is performed only on reference 0; only cost center 0 and start 0 are used. There is only one record. Adaptive search is also allowed. However, when AdaptiveEn is on, LenSU must be at least 2 as the adaptive search in VME is one-step delayed.
This is the common single directional motion search mode.</td></tr>
</table></td></tr>
</table>

DWord	Bits	Description
		001b: Single reference, single record and dual start. Search is performed only on reference 0; only cost center 0 is used. There is only one record. Search performs first on start 0 and then on start 1. Then if LenSP is not reached, the predetermined search path will start on start 1 with increment added to start 1 location. It then is followed by adaptive search. This is used for single direction adaptive search.
		011b: Single reference, dual record (and implied dual start). Search is performed only on reference 0; both cost center 0 and 1 and start 0 and 1 are used. Two records are used for both paths during IME. When integer search is complete, the two records are combined to find the best search. Sub-pel refinement is only performed from the best one. This may be used for search for multiple motion search candidates/predicators.
		111b: Dual reference, dual record (and implied dual start). Search is performed on references 0/1 with both cost centers 0/1 and starts 0/1. Two records are used for both paths during IME. When integer search is complete, and then sub-pel refinement is also performed separately, the two records are combined to find the best search on a subblock basis. This may be used for bidirectional motion search, or multi-reference P search. Whether bidirectional is enabled or not depends on the bidirection sub-macroblock mask. If BiSubMbPartMask is set to 1111'b, bidirectional search is disabled. VME outputs only the best unidirectional search results. Otherwise, BME is performed. Note that bidirectional search and sub-pel refinement are orthogonal features that can be enabled independently.
	7	**Reference Access (RefAccess)** This field defines how the reference blocks are accessed from the reference frames. It indicates if the source picture is a frame picture or a field picture. Programming Note: For all known video coding standards, reference pictures always have the same picture type as the source picture. Therefore, this field should be programmed to be the same as SrcAccess. 0: Frame based 1: Field based ValidMsgType = SIC, IME, FBR, IDM

DWord	Bits	Description
	6	**Source Access (SrcAccess)** This field defines how the source block is accessed from the source frame. It indicates if the source picture is a frame picture or a field picture. It is similar to the Picture Type used in video standards. 0: Frame based 1: Field based ValidMsgType = SIC, IME, FBR, IDM
	5:4	Inter MbType Remap (MbTypeRemap): This field controls the mapping of the output MbType when the VME output is an Inter (IntraMbFlag = INTER). The intended usage, for example, is for two forward (or backward) references or for two search regions from the same reference picture in one VME call. Hardware ignores this field if the VME output is an intra type (IntraMbFlag = INTRA). 00b: No remapping 01b: Remapping MbType to forward only (1-3 mapped to 1, even numbers in [4-14h] mapped to 4, odd numbers in [5-15h] mapped to 5, and 16h is unchanged) 10b: Remapping MbType to backward only (1-3 mapped to 2, even numbers in [4-14h] mapped to 6, odd numbers in [5-15h] mapped to 7, and 16h is unchanged) 11b: Reserved ValidMsgType = IME, FBR
	3	Reserved: MBZ
	2	Reserved: MBZ
	1:0	**Source Block Size (SrcSize)** This field defines how the 16x16 source block is formed. When Source Block Size is less than 16x16, SU larger than 4x4 is used. 00b: 16x16 01b: 16x8 10b: Reserved (for 8x16) 11b: 8x8 ValidMsgType = SIC, IME, FBR, IDM Note: For IDM message, the source block size should be always programmed to 16x16.

DWord	Bits	Description
M0.2	31:16	**Source Y (SrcY)** This field defines the vertical position (of the block's upper-left pixel) in units of pixels for the source block in the source frame. **Restriction** The Y address restriction is removed. Exception: for SIC messages where Intra Compute Type is set to 00 (Luma + Chroma enabled), SrcY must be a multiple of 2. ValidMsgType = SIC, IME, FBR, IDM Format = U16
	15:0	**Source X (SrcX)** This field defines the horizontal position (of the block's upper-left pixel) in units of pixels for the source block in the source picture. The source block must be within the source picture starting at any integer grid. For SIC messages where Intra Compute Type is set to 00 (Luma + Chroma enabled), SrcX must be a multiple of 2. ValidMsgType = SIC, IME, FBR, IDM Format = U16
M0.1	31:16	**Reference 1 Y Delta (Ref1Y)** This field defines the vertical position (of the upper-left corner of the reference region) in units of pixels for the Reference 1 region relative to the surface origin. **Restriction** The Y address restriction is removed. ValidMsgType = IME Format = S15 Hardware Range: [-2048 to 2047] Format = U16
	15:0	**Reference 1 X Delta (Ref1X)** This field defines the horizontal position (of the upper-left corner of the reference region) in units of pixels for the Reference 1 region relative to the surface origin. The resulting reference region is allowed to be outside the picture. Pixel replication is applied to

DWord	Bits	Description
		generate out of bound reference pixels. This field is only valid when dual reference mode is selected. Note: For search control=3, this must equal Ref0X. ValidMsgType = IME Format = S15 Hardware Range: [-2048 to 2047]
M0.0	31:16	**Reference 0 Y Delta (Ref0Y)** This field defines the vertical position (of the upper-left corner of the reference region) in units of pixels for the Reference 0 region relative to the surface origin. **Restriction** The Y address restriction is removed. ValidMsgType = IME, IDM Format = S15 Hardware Range: [-2048 to 2047] Format = U16
	15:0	**Reference 0 X Delta (Ref0X)** This field defines the horizontal position (of the upper-left corner of the reference region) in units of pixels for Reference 0 region relative to the surface origin. The resulting reference region is allowed to be outside the picture. Pixel replication is applied to generate out of bound reference pixels. ValidMsgType = IME, IDM Format = S15 Hardware Range: [-2048 to 2047]
M1.7	31:24	**Skip Center Enable Mask (SkipCenterMask):** [bits 31...24] xxxx xxx1: Ref0 Skip Center 0 is enabled [corresponds to M2.0] xxxx xx1x: Ref1 Skip Center 0 is enabled [corresponds to M2.1] xxxx x1xx: Ref0 Skip Center 1 is enabled [corresponds to M2.2] xxxx 1xxx: Ref1 Skip Center 1 is enabled [corresponds to M2.3] xxx1 xxxx: Ref0 Skip Center 2 is enabled [corresponds to M2.4]

DWord	Bits	Description
		xx1x xxxx: Ref1 Skip Center 2 is enabled [corresponds to M2.5] x1xx xxxx: Ref0 Skip Center 3 is enabled [corresponds to M2.6] 1xxx xxxx: Ref1 Skip Center 3 is enabled [corresponds to M2.7] Illegal cases: Disable both Ref0 and Ref1 Skip Center 0 in case of Skip_1MVP. Disable both Ref0 and Ref1 for any Skip Center pair in case of Skip_4MVP. ValidMsgType = SIC ValidMsgType = SIC
	23	IDM Shape Mode Select (IDMShapeMode): [Also see M1.1 bits 30 and 31] This bit selects what shape size the IDM is searching for. 0: 16x16 1: 8x8 ValidMsgType = IDM Note: Only ref window size of 32x32 (shape16x16), 24x24(shape8x8). Search control[2:0] defaults to single ref and single start. Luma only. 128x16 (shape16x16), and 32x16 (shape16x16) are supported. This bit must be set to 0 (i.e. shape 16x6) if it is IDM message type with ref window size of 128x16 and 32x16.
	22	**RefID Cost Mode Select (RefIDCostMode)** Selects the RefID costing mode. 0 = Mode0 (AVC) 1 = Mode1 (linear) ValidMsgType = SIC, IME, FBR, IDM
	21	**Enable AC-Only HAAR (AConlyHAAR)** This bit zeros out the DC component in the HAAR SATD block. 0 = AC+DC HAAR 1 = AC HAAR ValidMsgType = SIC, IME, FBR
	20	**Enable Weighted-SAD\HAAR (WeightedSADHAAR)** **Restriction** This bit enables weighted SAD\HAAR. 0: No weighted-SAD

DWord	Bits	Description
		1: Enable weighted-SAD Note: if this bit is 1, ShapeMask is ignored and only 16x16 shapes are accumulated (no partitioning). Restrictions: Only supported for source-type luma 16x16. Only support unidirectional search (on Ref0). See M1.3 for individual sub-block weight control. ValidMsgType = IME, IDM
	19	**Source Field Polarity Select (SrcFieldPolarity)** If SrcAccess = 1 (M0.3-6), meaning field based, than the hardware requires this value to derive the correct location on the source surface in memory to fetch pixels. This is because the source is stored as a frame picture with both fields interleaved in memory and the SrcY value (M0.2-31:16) is the location on the field picture (in other words, it does not convey the field polarity). Hence, the starting y-pixel coordinate fetched from memory is: SrcY* 2 + SrcFieldPolarity Else, this field is ignored by the hardware. ValidMsgType = SIC, IME, FBR, IDM Format = U1
	18	**Bilinear Filter Enable (BilinearEnable)** If set, the fractional filter implements a simple bilinear interpolation filter instead of the 4-tap filter. Note: This is supported for both hpel and qpel interpolation. ValidMsgType = SIC, FBR Format = Enable
	17:16	**MV Cost Scaling Factor (MVCostScaleFactor)** This term allows the user to redefine the precision of the lookup into the LUT_MV based on the MV cost difference from the cost center. The piecewise linear cost function is defined from 0 to 64 in powers of 2 intervals, and the precision of the difference is set by this field. There are 4 precision choices: 00b: Qpel [Qpel difference between MV and cost center: eff cost range 0-15pel] 01b: Hpel [Hpel difference between MV and cost center: eff cost range 0-31pel] 10b: Pel [Pel difference between MV and cost center: eff cost range 0-63pel] 11b: 2pel [2Pel difference between MV and cost center: eff cost range 0-127pel] ValidMsgType = SIC, IME, FBR, IDM

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>Format = U2</td></tr>
<tr><td></td><td>15:8</td><td>Macroblock Intra Structure (MbIntraStruct): This is a bitmask that specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search.

Note: The user must set Bit6=Bit5.

The bit positions in the following table are relative positions within the field. For example, bit 7 in the table is bit 15 in the containing DWord.

<table><tr><th>Bit</th><th>MotionVerticalFieldSelect Index</th></tr><tr><td>7</td><td>Reserved: MBZ (for IntraPredAvailFlagF – F (pixel[-1,7] available for MbAff)</td></tr><tr><td>6</td><td>Reserved: MBZ (for IntraPredAvailFlagA/E – A (left neighbor top half for MbAff)</td></tr><tr><td>5</td><td>IntraPredAvailFlagE/A – A (Left neighbor or Left bottom half)</td></tr><tr><td>4</td><td>IntraPredAvailFlagB – B (Upper neighbor)</td></tr><tr><td>3</td><td>IntraPredAvailFlagC – C (Upper left neighbor)</td></tr><tr><td>2</td><td>IntraPredAvailFlagD – D (Upper right neighbor)</td></tr><tr><td>1:0</td><td>Reserved: MBZ (ChromaIntraPredMode)</td></tr></table>
ValidMsgType = SIC</td></tr>
<tr><td></td><td>7</td><td>Luma Intra Source Corner Swap (IntraCornerSwap): This field specifies the format of the intra luma neighbor pixel format in the message.

0: Top neighbors are in sequential order.

1: Left-top corner is swapped with the last left-edge neighbor.

ValidMsgType = SIC</td></tr>
<tr><td></td><td>6</td><td>Non Skip MB Mode Cost Added (NonSkipModeAdded)

This field indicates that the distortion of the survived motion vectors becomes non-skip, and the MB mode cost is added to its distortion.

ValidMsgType = SIC</td></tr>
<tr><td></td><td>5</td><td>Non Skip Zero MV Cost Added (NonSkipZMvAdded)

This field indicates that the distortion of the survived motion vectors becomes non-skip, and the zero MV component costs are added to its distortion.

ValidMsgType = SIC</td></tr>
<tr><td></td><td>4:0</td><td>Luma Intra Partition Mask (IntraPartMask)

This field specifies which Luma Intra partition is enabled/disabled for intra mode decision.

xxxx1: luma_intra_16x16 disabled</td></tr>
</table>

DWord	Bits	Description
		xxx1x: luma_intra_8x8 disabled xx1xx: luma_intra_4x4 disabled Note: For SIC message with IntraComputeType == 00 or 01, at least 1 partition must be enabled. Bits [4:3] MBZ ValidMsgType = SIC
M1.6	31:28	**Bwd Block 3 RefID**
	27:24	**Fwd Block 3 RefID**
	23:20	**Bwd Block 2 RefID**
	19:16	**Fwd Block 2 RefID**
	15:12	**Bwd Block 1 RefID**
	11:8	**Fwd Block 1 RefID**
	7:4	**Bwd Block 0 RefID**
	3:0	**Fwd Block 0 RefID** M1.6 contains 8 input RefIDs, 1 per block. The RefID is used to penalize selection of shapes away from the optimal RefID similar to how MVCost penalizes shapes with motion vectors far from the cost center. **Restriction** Note: All 4 Bwd RefID are ignored by HW for IDM message type. Format = U4 ValidMsgType = SIC, IME, FBR, IDM
M1.5	31:16	Reserved: MBZ
	15:0	Reserved: MBZ
M1.4	31:16	Reserved: MBZ
	15:0	Reserved: MBZ
M1.3	31:30	**Weighted SAD Control Sub-block 15 (F)**
	29:28	**Weighted SAD Control Sub-block 14 (E)**
	27:26	**Weighted SAD Control Sub-block 13 (D)**

DWord	Bits	Description
	25:24	**Weighted SAD Control Sub-block 12 (C)**
	23:22	**Weighted SAD Control Sub-block 11 (B)**
	21:20	**Weighted SAD Control Sub-block 10 (A)**
	19:18	**Weighted SAD Control Sub-block 9**
	17:16	**Weighted SAD Control Sub-block 8**
	15:14	**Weighted SAD Control Sub-block 7**
	13:12	**Weighted SAD Control Sub-block 6**
	11:10	**Weighted SAD Control Sub-block 5**
	9:8	**Weighted SAD Control Sub-block 4**
	7:6	**Weighted SAD Control Sub-block 3**
	5:4	**Weighted SAD Control Sub-block 2**
	3:2	**Weighted SAD Control Sub-block 1**
	1:0	**Weighted SAD Control Sub-block 0** When the Weighted SAD control is enabled (M1.7 bit 20) these values are used to decrease the magnitude of each sub-block by dividing the 4x4 SAD\HAAR output mapped to that 4x4 of the source MB. The control value divides the 4x4 SAD\HAAR output by 2^control value. 0: »0 (div by 1) 1: »1 (div by 2) 2: »2 (div by 4) 3: »3 (div by 8) The output produces 1 16x16 macroblock results each with different weighted-SAD control. The values from M1.3 31:0 are mapped onto the sub-blocks of the source MB in the traditional Z-order: 16x16_0 Weighted-SAD Control Mapping: 0145 2367 89CD ABEF

DWord	Bits	Description
		ValidMsgType = IME, IDM
M1.2	31:28	**Start Center 1 Y (Start1Y)** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in units of SU. ValidMsgType = IME Format = U4
	27:24	**Start Center 1 (Start1X)** This field defines the X position of Search Path 1 relative to the reference X location. It is in units of SU. The corresponding reference block must be fully within the reference region. ValidMsgType = IME Format = U4
	23:20	**Start Center 0 Y (Start0Y)** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in units of SU. ValidMsgType = IME Format = U4
	19:16	**Start Center 0 X (Start0X)** This field defines the X position of Search Path 1 relative to the reference X location. It is in units of SU. The corresponding reference block must be fully within the reference region. ValidMsgType = IME Format = U4
	15:8	**Maximum Search Path Length (MaxNumSU)** This field defines the maximum number of SUs per reference including the predetermined SUs and the adaptively generated SUs. Note: Every SU in fixed path is counted (including the out-bound ones and repeated ones), and in addition for adaptive SUs only the ones actually searched are added. ValidMsgType = IME Format = U8, with valid range of [1,63]

DWord	Bits	Description
	7:0	**Max Fixed Search Path Length (LenSP)** This field defines the maximum number of SUs per reference which are evaluated by the predetermined SUs. When adaptive walk is enabled, adaptive walk starts when this number is reached. Note: Every SU in fixed path is counted (including the out-bound ones and repeated ones). ValidMsgType = IME Format = U8, with valid range of [1,63]
M1.1	31	**IDM Shape Mode 5x5 Select** (IDMShapeMode5x5) Extension This bit selects what shape size the IDM is searching for. For CenterWin, only 5x5 filter is supported. It is illegal to turn on both 5x5 and 7x7 filter at the same time. 0: Other 1: 5x5 ValidMsgType = IDM
	30	**IDM Shape Mode 7x7 Select (IDMShapeMode7x7) Extension** This bit selects what shape size the IDM is searching for. For SadofSobels, only 7x7 filter is supported. It is illegal to turn on both 5x5 and 7x7 filter at the same time. 0: Other 1: 7x7 ValidMsgType = IDM Note: The Source Block Size needs to be programmed to 16x8 if this is enabled.
	29	Reserved: MBZ
	28	Unidirectional Mix Disable (UniMixDisable): If it is on, all unidirectional resulting motion vectors must share the same direction, i.e. either all are forward, or all are backward. If this field is off, each partition, down to 8x8 subblock, may have a different mix of forward and backward motion vectors. (Within each 8x8 subblock, only one common choice is allowed.) Programming Note: For the case when BMEdisableFBR is set, only the input subpredmode direction is refined. If BMEdisableFBR is not set, both directions undergo fractional refinement before bidirectional refinement, but the subpredmode output never inverts directions if the refinement yields a better result (subpredmode could change to bidirectional in this case though). This field is MBZ except for cases of Search Control = 111'b (e.g. 7, dual reference). ValidMsgType = IME

DWord	Bits	Description
	27:24	RefPixelShift The amount the reference pixels are shifted positive or negative. The MSB implies right shift if zero or left shift if positive. The other three bits specify the amount that is shifted.
	23:22	Reserved: MBZ [Fixed7x7Weights -- GxMask, GyMask]
	21:16	**Bidirectional Weight (BiWeight)** This field defines the weighting for the backward and forward terms to generate the bidirectional term. This field is only valid for bidirectional search (SearchCtrl = 111). ValidMsgType = SIC, FBR Format = U6 Valid Values: [16, 21, 32, 43, 48]
	15:8	**RefId Polarity Bits** Bit15->bwd block3 Bit14->bwd block2 Bit13->bwd block1 Bit12->bwd block0 Bit11->fwd block3 Bit10->fwd block2 Bit9->fwd block1 Bit8->fwd block0 ValidMsg type - IME, SIC, FBR, IDM
	7	Reserved: MBZ
	6	Extended MV Cost Range This bit specifies if the increased 12-bit mvcost range is used vs. the legacy 10-bit range. 0 = Disable 1 = Enable ValidMsgType = SIC,IME, FBR
	5:0	**Maximum Number of Motion Vectors (MaxNumMVs)** This field specifies the maximum number of motion vectors allowed for the current macroblock. This field affects the macroblock partition decision. VME returns the best partition with MvQuantity

DWord	Bits	Description
		not exceeding MaxNumMVs. MaxNumMVs = 0 only allows skip as a valid Inter mode. Note: This value is used ONLY for 16x16 source MB mode. Usage Example: Certain profiles/levels for AVC standard have restrictions for the maximum number of motion vectors allowed for two consecutive macroblocks (MaxMvsPer2Mb may be 16 or 32). ValidMsgType = IME Format = U6
M1.0	31:24	**Early IME Successful Stop Threshold (EarlyImeStop)** This field specifies the threshold value for the IME distortion computes of single 16x16 mode below which no more search is performed within the IME unit. This field only takes effect if EarlyImeSuccessEn is set. Note: Early IME exit only looks at ref0, and uses 8x8 for source 8x8 and 2 16x8 0 for source 16x8. ValidMsgType = IME Format = U4U4 (encoded value should fit in 14 bits)
	23:16	Reserved: MBZ
	15:8	RefPixelOffset The amount the reference pixels are offset. This is a signed 8 bits so [-128,127].
	7	**Transform 8x8 Flag For Inter Enable (T8x8FlagForInterEn)** This field specifies whether Transform8x8Flag is updated for inter mode according to the resulting inter-mode sub-partition size. 0: Disable 1: Enable ValidMsgType = SIC, IME, FBR
	6	**X only search** This field enables searching in only the x dimension. ValidMsg type - IME, IDM X-only search supports only Lumma and non-adaptive search. This bit needs to be set to 0 for 1d-sobel and center-win.
	5	**Early IME Success Enable (EarlyImeSuccessEn)** This field specifies whether the Early Success may terminate on full-pel precision. When this field is not set, if early out does occur on full-pel location, hardware continues to local sub-pel refinement

DWord	Bits	Description
		search and so on. When this field is set, however, the local sub-pel refinement step is skipped and intra search is also skipped. This field only takes effect if EarlySuccessEn is set. Usage Example: This may be used for cases with large static area where (0,0) motion vector delivers very good results that no FME refinement is needed and also intra check is also skipped. This may also be used in place of Skip Mode Checking when the skip center(s) happens to be an integer location inside the SU of the Start Center(s). 0: Disable 1: Enable ValidMsgType = IME
	4:3	Reserved: MBZ
	2	Bidirectional Mix Disable (BiMixDis): If it is on, all resulting motion vectors must share the same direction, i.e. either all are unidirectional (i.e. forward or backward), or all bidirectional. If this field is off, each partition may have different search direction (forward, backward, or bidirectional). Usage Example: MPEG2 bidirectional decision is at whole macroblock level, while AVC decision is at subblock level. 0: Bidirectional decision on subblock level that bidirectional mode is enabled. 1: Bidirectional decision on whole macroblock. Note: This must be disabled for SubMbShape with any minors (8x4/4x8/4x4) in the MB. ValidMsgType = FBR
	1	Adaptive Search Enable (AdaptiveEn): This field defines whether adaptive searching is enabled for IME. When Adaptive Search is enabled, there must be at least two search steps preceded. It is either from a single start with step of >=2 or from a dual-start. 0: Disable 1: Enable ValidMsgType = IME
	0	Skip Mode Enable (SkipModeEn): This field specifies whether the skip mode checking is performed before the motion search. If this field is set, Skip Center, which may have a sub-pel precision, is first tested before IME. 0: Disable 1: Enable
M2.7	31:24	**SIC Forward Transform Coeff Threshold Matrix[6] - Highest Freq**

DWord	Bits	Description
	23:16	**SIC Forward Transform Coeff Threshold Matrix[5]**
	15:8	**SIC Forward Transform Coeff Threshold Matrix[4]**
	7:0	**SIC Forward Transform Coeff Threshold Matrix[3]**
M2.6	31:24	**SIC Forward Transform Coeff Threshold Matrix[2]**
	23:16	**SIC Forward Transform Coeff Threshold Matrix[1]**
	15:0	**SIC Forward Transform Coeff Threshold Matrix[0]** Values of the threshold matrix[0..6] are provided here. Matrix[0] contains the DC threashold for the Forward Transform Skip check. It has increased precision vs. the other thresholds due to the larger size of DC coeffieicients. Matrix[1] through Matrix[6] have lower precision. Threshold Matrix for 4x4 transform is as follows: 0 1 2 3 1 2 3 4 2 3 4 5 3 4 5 6 Matrix[0] Format = U16 Matrix[1..6] Format = U8 ValidMsgType = SIC
M2.5	31:24	Reserved: MBZ
	23:16	**FBR SubPredMode Input** VME uses this to select the appropriate shapes from the input message to perform FME on. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3] 00: Forward 01: Backward 10: Bidirectional 11: Illegal

DWord	Bits	Description
		Note: Only the LSB of the subpredmode for each shape is considered in FBR (a shape is either FWD or BWD as input of FBR). ValidMsgType = FBR
	15:8	**FBR SubMBShape Input** This field is used to specify the subshape per block for fractional and bidirectional refinement. Bits [1:0]: SubMbShape[0] Bits [3:2]: SubMbShape[1] Bits [5:4]: SubMbShape[2] Bits [7:6]: SubMbShape[3] 00: 8x8 01: 8x4 10: 4x8 11: 4x4 ValidMsgType = FBR
	7:2	Reserved: MBZ
	1:0	**FBR MbMode Input** This field is used to specify the inter macroblock type in the same format as VME output. 00: 16x16 01: 16x8 10: 8x16 11: 8x8 ValidMsgType = FBR
M2.4	31:24	**MV 7 Cost**
	23:16	**MV 6 Cost**
	15:8	**MV 5 Cost**
	7:0	**MV 4 Cost**
M2.3	31:24	**MV 3 Cost**
	23:16	**MV 2 Cost**

DWord	Bits	Description
	15:8	**MV 1 Cost**
	7:0	**MV 0 Cost** Motion vector costings. See 6.3.3.1 for details. In short, the cost is linearly interpolated between control points. Format = U4U4 (encoded value must fit in 10 bits) **Note** ValidMsgType = SIC, IME, FBR, IDM
M2.2	31:24	**Chroma Intra Mode Cost** Penalty for chroma intra modes. DC = 0x Horz = 1x Vert = 1x Plane = 2x Format = U4U4 (encoded value must fit in 12-bits) ValidMsgType = SIC, IME, FBR
	23:16	**RefID Cost** RefID costing base penalty. Under AVC or Linear mode, different scaling are applied on top of this. Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR, IDM
	15:8	**Mode 9 Cost** MODE_INTER_BWD Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	7:0	**Mode 8 Cost** MODE_INTER_16x16 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR, IDM
M2.1	31:24	**Mode 7 Cost** MODE_INTER_4x4q

DWord	Bits	Description
		MODE_INTER_FIELD_8x8q Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
	23:16	**Mode 6 Cost** MODE_INTER_8x4q MODE_INTER_4x8q MODE_INTER_FIELD_16x8 Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
	15:8	**Mode 5 Cost** MODE_INTER_8x8q Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR, IDM
	7:0	**Mode 4 Cost** MODE_INTER_16x8 MODE_INTER_8x16 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
M2.0	31:24	**Mode 3 Cost** MODE_INTRA_4x4 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	23:16	**Mode 2 Cost** MODE_INTRA_8x8 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	15:8	**Mode 1 Cost** MODE_INTRA_16x16 Format = U4U4 (encoded value must fit in 12 bits)

DWord	Bits	Description
		ValidMsgType = SIC, IME, FBR
	7:0	**Mode 0 Cost** MODE_INTRA_NONPRED Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
M3.7	31:0	**BWD Cost Center 3**
M3.6	31:0	**FWD Cost Center 3**
M3.5	31:0	**BWD Cost Center 2**
M3.4	31:0	**FWD Cost Center 2**
M3.3	31:0	**BWD Cost Center 1**
M3.2	31:0	**FWD Cost Center 1**
M3.1	31:16	**BWD Cost Center 0 Delta Y (BWDCostCenter0Y)** This field defines the Y value for the first cost center relative to the picture source MB Y value for the BWD direction. All 4 Bwd Cost Center Deltas are ignored by HW for IDM message type. ValidMsgType = SIC, IME, FBR Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	**BWD Cost Center 0 Delta X (BWDCostCenter0X)** This field defines the X value for the first cost center relative to the picture source MB X value for the BWD direction. Major shape mapping to each cost center: CC0: 16x16_0, 16x8_0, 8x16_0, 8x8_0 CC1: 8x16_1, 8x8_1 CC2: 16x8_1, 8x8_2 CC3: 8x8_3

DWord	Bits	Description
		All 4 Bwd Cost Center Deltas are ignored by HW for IDM message type. ValidMsgType = SIC, IME, FBR Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
M3.0	31:16	FWDCostCenter 0 Delta Y (FWDCostCenter0Y): This field defines the Y value for the first cost center relative to the picture source MB Y value for the FWD direction. ValidMsgType = SIC, IME, FBR, IDM Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	FWDCostCenter 0 Delta X (FWDCostCenter0X): This field defines the X value for the first cost center relative to the picture source MB X value for the FWD direction. Major shape mapping to each cost center: CC0: 16x16_0, 16x8_0, 8x16_0, 8x8_0 CC1: 8x16_1, 8x8_1 CC2: 16x8_1, 8x8_2 CC3: 8x8_3 ValidMsgType = SIC, IME, FBR, IDM Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]

SIC Input Message Phases

Major changes

- Addition of chroma pixel pairs (CbCr as 16b value) for the left 8, top 8, and top-left 1 corner.
- Addition of chroma mode masks (only 4 modes possible, so 4b mask).
- Addition of intra compute type (Y+CbCr, Y only, disabled).

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

DWord	Bits	Name
WX+0.7	31:0	**Ref1 SkipCenter 3 Delta XY** Ref1 Skip Center 3 Delta Y: This field defines the Y value for the forward skip center relative to the 8x8 block offset from the source MB Y location in quarter-pel precision associated with Ref1. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input Y value. For SkipCenter 1: VME will add 0 to the user-input Y value. For SkipCenter 2: VME will add 32 to the user-input Y value. For SkipCenter 3: VME will add 32 to the user-input Y value. ValidMsgType = SIC Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75] For chroma skip: Format = S12.3 (2's comp) Hardware Range: [-256.000 to 255.875] Ref1SkipCenter3 Delta X: This field defines the X value for the forward skip center relative to the 8x8 block offset from the source MB X location in quarter-pel precision associated with Ref1. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input X value. For SkipCenter 1: VME will add 32 to the user-input X value. For SkipCenter 2: VME will add 0 to the user-input X value. For SkipCenter 3: VME will add 32 to the user-input X value. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] For chroma skip: Format = S12.3 (2's comp) Hardware Range: [-1024.000 to 1023.875]
WX+0.6	31:0	**Ref0 SkipCenter 3 Delta XY***(for definition see M3.7)*

DWord	Bits	Name
WX+0.5	31:0	**Ref1 SkipCenter 2 Delta XY** *(for definition see M3.7)*
WX+0.4	31:0	**Ref0 SkipCenter 2 Delta XY** *(for definition see M3.7)*
WX+0.3	31:0	**Ref1 SkipCenter 1 Delta XY** *(for definition see M3.7)*
WX+0.2	31:0	**Ref0 SkipCenter 1 Delta XY** *(for definition see M3.7)*
WX+0.1	31:0	**Ref1 SkipCenter 0 Delta XY** *(for definition see M3.7)*
WX+0.0	31:0	**Ref0 SkipCenter 0 Delta XY** *(for definition see M3.7)*
WX+1.7	31:0	**Neighbor pixel Luma value [23, -1] to [20, -1].** Upper-right pixels from neighbor macroblock C
WX+1.6	31:0	**Neighbor pixel Luma value [19, -1] to [16, -1].** Upper-right edge pixels from neighbor macroblock C
WX+1.5	31:0	**Neighbor pixel Luma value [15, -1] to [12, -1].** Top edge pixels from neighbor macroblock B
WX+1.4	31:0	**Neighbor pixel Luma value [11, -1] to [8, -1].** Top edge pixels from neighbor macroblock B
WX+1.3	31:0	**Neighbor pixel Luma value [7, -1] to [4, -1].** Top edge pixels from neighbor macroblock B
WX+1.2	31:24	**Neighbor pixel Luma value [3, -1].** Fourth top edge pixel from neighbor macroblock B
	23:16	**Neighbor pixel Luma value [2, -1].** Third top edge pixel from neighbor macroblock B
	15:8	**Neighbor pixel Luma value [1, -1].** Second top edge pixel from neighbor macroblock B
	7:0	**Neighbor pixel Luma value [0, -1].** First top edge pixel from neighbor macroblock B
WX+1.1	31:24	**Corner Neighbor pixel 0.** Its content depends on IntraCornerSwap field. It swaps with Corner Neighbor pixel 1.
	23:10	Reserved: MBZ
	9:8	**Intra Compute Type (IntraComputeType)** This field specifies the pixel components measured for intra prediction. 00: Luma + Chroma enabled 01: Luma only

DWord	Bits	Name
		1X: Intra disabled
	7:4	**AVC Intra Chroma Mode Mask (IntraChromaModeMask)** The following mask disables the chroma intra modes from the output. xxx1: VERT xx1x: HORZ x1xx: DC 1xxx: PLANAR
	3:0	**AVC Intra 16x16 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. xxx1: xx1x: x1xx: 1xxx:
WX+1.0	31:25	Reserved: MBZ
	24:16	**AVC Intra 8x8 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:
	15:9	Reserved: MBZ
	8:0	**AVC Intra 4x4 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows.

DWord	Bits	Name
		x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:
WX+2.7	31:24	Reserved: MBZ
	23:16	Penalty for Intra4x4 non-DC prediction mode Format: U8
	15:8	Penalty for Intra8x8 non-DC prediction mode Format: U8
	7:0	Penalty for Intra16x16 non-DC prediction mode Format: U8
WX+2.6	31:0	Reserved: MBZ
WX+2.5	31:16	Reserved: MBZ
	15:0	**Neighbor pixel Chroma value CbCr pair [-1, -1]** Corner neighbor pixel pair (CbCr pair, each U8).
WX+2.4	31:28	**Intra Predictor Mode for Neighbor B15 (IntraMxMPredModeB15):** This field carries the intra prediction mode of the fourth bottom 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	27:24	**Intra Predictor Mode for Neighbor B14 (IntraMxMPredModeB14):** This field carries the intra prediction mode of the third bottom 4x4 block (Block 14 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	23:20	**Intra Predictor Mode for Neighbor B11 (IntraMxMPredModeB11):** This field carries the intra prediction mode of the second bottom 4x4 block (Block 11 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and

DWord	Bits	Name
		8.3.2 of the AVC specification.
	19:16	**Intra Predictor Mode for Neighbor B10 (IntraMxMPredModeB10):** This field carries the intra prediction mode of the first bottom 4x4 block (Block 10 in Numbers of Block4x4 in a 16x16 region)of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	15:12	**Intra Predictor Mode for Neighbor A15 (IntraMxMPredModeA15):** This field carries the intra prediction mode of the fourth rightmost 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	11:8	**Intra Predictor Mode for Neighbor A13 (IntraMxMPredModeA13):** This field carries the intra prediction mode of the third rightmost 4x4 block (Block 13 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	7:4	**Intra Predictor Mode for Neighbor A7 (IntraMxMPredModeA7):** This field carries the intra prediction mode of the second rightmost 4x4 block (Block 7 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A.
	3:0	**Intra Predictor Mode for Neighbor A5 (IntraMxMPredModeA5):** This field carries the intra prediction mode of the first rightmost 4x4 block (Block 5 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification. Intra Predictor Modes for Neighbor A and B are only used if **MODE_INTRA_NOPRED** is not zero. For intra mode selection, bias is applied to the predicted mode if a predictor is present for a partition. This is achieved by applying a penalty term MODE_INTRA_NONPRED defined in the VME state to the cost functions for non-predicted modes. The predictor for a given partition is from its left neighbor and top neighbor. The intra decision for a partition serves as the predictor for the next partition in the partition order as defined in Numbers of Block4x4 in a 16x16 region and Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region. This set of intra predictor mode for neighbor macroblocks are only used for INTRA8x8 and INTRA4x4 modes. Format : U4 (The value of this field is defined in Definition of Intra4x4PredMode which is the same as that in Definition of Intra8x8PredMode.)
WX+2.3	31:24	**Corner Neighbor pixel 1.** Its content depends on **IntraCornerSwap** field. It swaps with **Corner Neighbor pixel 0**. **Neighbor pixel Luma value [-1, -1]**. The one upper-left edge pixel from neighbor macroblock D, which is the right most edge pixel of D, if **IntraCornerSwap** field is **1**. Or

DWord	Bits	Name
		Neighbor pixel Luma value [-1, 15]. The last left edge pixel from neighbor macroblock A, which is the left most edge pixel of D, if **IntraCornerSwap** field is 0.
	23:0	**Neighbor pixel Luma value [-1, 14] to [-1, 12].** Left edge pixels from neighbor macroblock A
WX+2.2	31:0	**Neighbor pixel Luma value [-1, 11] to [-1, 8].** Left edge pixels from neighbor macroblock A
WX+2.1	31:0	**Neighbor pixel Luma value [-1, 7] to [-1, 4].** Left edge pixels from neighbor macroblock A
WX+2.0	31:24	**Neighbor pixel Luma value [-1, 3].** Fourth left edge pixel from neighbor macroblock A
	23:16	**Neighbor pixel Luma value [-1, 2].** Third left edge pixel from neighbor macroblock A
	15:8	**Neighbor pixel Luma value [-1, 1].** Second left edge pixel from neighbor macroblock A
	7:0	**Neighbor pixel Luma value [-1, 0].** First left edge pixel from neighbor macroblock A
WX+3.7	31:0	**Neighbor pixel Chroma value CbCr pair [7, -1] to [6, -1]**
WX+3.6	31:0	**Neighbor pixel Chroma value CbCr pair [5, -1] to [4, -1]**
WX+3.5	31:0	**Neighbor pixel Chroma value CbCr pair [3, -1] to [2, -1]**
WX+3.4	31:0	**Neighbor pixel Chroma value CbCr pair [1, -1] to [0, -1]**
WX+3.3	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 7] to [-1, 6]**
WX+3.2	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 5] to [-1, 4]**
WX+3.1	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 3] to [-1, 2]**
WX+3.0	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 1] to [-1, 0]**

IME Input Message Phases

Major changes:

- Addition of the search path, no longer accessed via LUT, will come in message payload.
- Streamin\streamout now contains the 9 major shape reference indices per direction.
- Distortion precisions increased to 16b.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

DWord	Bits	Name
WX+0.7	31:0	**IME Search Path Delta 28-31**
WX+0.6	31:0	**IME Search Path Delta 24-27**
WX+0.5	31:0	**IME Search Path Delta 20-23**
WX+0.4	31:0	**IME Search Path Delta 16-19**
WX+0.3	31:0	**IME Search Path Delta 12-15**
WX+0.2	31:0	**IME Search Path Delta 8-11**
WX+0.1	31:0	**IME Search Path Delta 4-7**
WX+0.0	31:0	**IME Search Path Delta 0-3** [7:4] (Y) – specifies relative Y distance to the next SU from previous SU in units of SU. [3:0] (X) – specifies relative X distance to the next SU from previous SU in units of SU. Format = U8
WX+1.7	31:0	Reserved MBZ
WX+1.6	31:0	Reserved MBZ
WX+1.5	31:0	**IME Search Path Delta 52-55**
WX+1.4	31:0	**IME Search Path Delta 48-51**
WX+1.3	31:0	**IME Search Path Delta 44-47**
WX+1.2	31:0	**IME Search Path Delta 40-43**
WX+1.1	31:0	**IME Search Path Delta 36-39**
WX+1.0	31:0	**IME Search Path Delta 32-35**
WX+2.7	31:0	Reserved MBZ
WX+2.6	31:28	Rec0 Shape 8x8_3 RefID
	27:24	Rec0 Shape 8x8_2 RefID

DWord	Bits	Name
	23:20	Rec0 Shape 8x8_1 RefID
	19:16	Rec0 Shape 8x8_0 RefID
	15:12	Rec0 Shape 8x16_1 RefID
	11:8	Rec0 Shape 8x16_0 RefID
	7:4	Rec0 Shape 16x8_1 RefID
	3:0	Rec0 Shape 16x8_0 RefID Format = U4
WX+2.5	31:16	Rec0 Shape 16x16 Y (relative to source MB)
	15:0	Rec0 Shape 16x16 X (relative to source MB)
WX+2.4	31:20	Reserved MBZ
	19:16	Rec0 Shape 16x16 RefID Format = U4
	15:0	Rec0 Shape 16x16 Distortion Format = U16
WX+2.3	31:16	Rec0 Shape 8x8_3 Distortion Format = U16 Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).
	15:0	Rec0 Shape 8x8_2 Distortion Format = U16
WX+2.2	31:16	Rec0 Shape 8x8_1 Distortion Format = U16
	15:0	Rec0 Shape 8x8_0 Distortion Format = U16
WX+2.1	31:16	Rec0 Shape 8x16_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec0 Shape 8x16_0 Distortion

DWord	Bits	Name
		Format = U16
WX+2.0	31:16	Rec0 Shape 16x8_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec0 Shape 16x8_0 Distortion Format = U16
WX+3.7	31:16	Rec0 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_3 X (relative to source MB)
WX+3.6	31:16	Rec0 Shape 8x8_2 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_2 X (relative to source MB)
WX+3.5	31:16	Rec0 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_1 X (relative to source MB)
WX+3.4	31:16	Rec0 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_0 X (relative to source MB)
WX+3.3	31:16	Rec0 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_1 X (relative to source MB)
WX+3.2	31:16	Rec0 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_0 X (relative to source MB)
WX+3.1	31:16	Rec0 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_1 X (relative to source MB)
WX+3.0	31:16	Rec0 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_0 X (relative to source MB)
WX+4.7	31:0	Reserved MBZ
WX+4.6	31:28	Rec1 Shape 8x8_3 RefID
	27:24	Rec1 Shape 8x8_2 RefID
	23:20	Rec1 Shape 8x8_1 RefID
	19:16	Rec1 Shape 8x8_0 RefID
	15:12	Rec1 Shape 8x16_1 RefID
	11:8	Rec1 Shape 8x16_0 RefID
	7:4	Rec1 Shape 16x8_1 RefID
	3:0	Rec1 Shape 16x8_0 RefID Format = U4

DWord	Bits	Name
WX+4.5	31:16	Rec1 Shape 16x16 Y (relative to source MB)
	15:0	Rec1 Shape 16x16 X (relative to source MB)
WX+4.4	31:20	Reserved MBZ
	19:16	Rec1 Shape 16x16 RefID Format = U4
	15:0	Rec1 Shape 16x16 Distortion Format = U16
WX+4.3	31:16	Rec1 Shape 8x8_3 Distortion Format = U16 Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).
	15:0	Rec1 Shape 8x8_2 Distortion Format = U16
WX+4.2	31:16	Rec1 Shape 8x8_1 Distortion Format = U16
	15:0	Rec1 Shape 8x8_0 Distortion Format = U16
WX+4.1	31:16	Rec1 Shape 8x16_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec1 Shape 8x16_0 Distortion Format = U16
WX+4.0	31:16	Rec1 Shape 16x8_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec1 Shape 16x8_0 Distortion Format = U16

DWord	Bits	Name
WX+5.7	31:16	Rec1 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_3 X (relative to source MB)
WX+5.6	31:16	Rec1 Shape 8x8_2 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_2 X (relative to source MB)
WX+5.5	31:16	Rec1 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_1 X (relative to source MB)
WX+5.4	31:16	Rec1 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_0 X (relative to source MB)
WX+5.3	31:16	Rec1 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_1 X (relative to source MB)
WX+5.2	31:16	Rec1 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_0 X (relative to source MB)
WX+5.1	31:16	Rec1 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_1 X (relative to source MB)
WX+5.0	31:16	Rec1 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_0 X (relative to source MB)

FBR Input Message Phases

Major changes:

- Consists of the 32 sub-block motion vectors following the same 32MV format as the rest of VME.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

DWord	Bits	Name
WX+0.7	31:0	**Ref1 Sub-block XY 3**
WX+0.6	31:0	**Ref0 Sub-block XY 3**
WX+0.5	31:0	**Ref1 Sub-block XY 2**
WX+0.4	31:0	**Ref0 Sub-block XY 2**
WX+0.3	31:0	**Ref1 Sub-block XY 1**
WX+0.2	31:0	**Ref0 Sub-block XY 1**

DWord	Bits	Name
WX+0.1	31:0	**Ref1 Sub-block XY 0**
WX+0.0	31:16	**Ref0 Sub-block Y 0** The y-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. **Note:** All MVs must be replicated for each shape. (e.g. for luma 16x16 shape and chroma 8x8, all Sub-block MVs must be the same. For luma 8x8 shape and chroma 4x4, each 8x8 must have its respective Sub-block MVs be replicated). Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
	15:0	**Ref0 Sub-block X 0** The x-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. **Note**: All MVs must be replicated for each shape. (e.g. for luma 16x16 shape and chroma 8x8, all Sub-block MVs must be the same. For luma 8x8 shape and chroma 4x4, each 8x8 must have its respective Sub-block MVs be replicated). Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
WX+1.7	31:0	**Ref1 Sub-block XY 7**
WX+1.6	31:0	**Ref0 Sub-block XY 7**
WX+1.5	31:0	**Ref1 Sub-block XY 6**
WX+1.4	31:0	**Ref0 Sub-block XY 6**
WX+1.3	31:0	**Ref1 Sub-block XY 5**
WX+1.2	31:0	**Ref0 Sub-block XY 5**
WX+1.1	31:0	**Ref1 Sub-block XY 4**
WX+1.0	31:0	**Ref0 Sub-block XY 4**
WX+2.7	31:0	**Ref1 Sub-block XY 11**
WX+2.6	31:0	**Ref0 Sub-block XY 11**
WX+2.5	31:0	**Ref1 Sub-block XY 10**

DWord	Bits	Name
WX+2.4	31:0	**Ref0 Sub-block XY 10**
WX+2.3	31:0	**Ref1 Sub-block XY 9**
WX+2.2	31:0	**Ref0 Sub-block XY 9**
WX+2.1	31:0	**Ref1 Sub-block XY 8**
WX+2.0	31:0	**Ref0 Sub-block XY 8**
WX+3.7	31:0	**Ref1 Sub-block XY 15**
WX+3.6	31:0	Ref0 Sub-block XY 15
WX+3.5	31:0	Ref1 Sub-block XY 14
WX+3.4	31:0	Ref0 Sub-block XY 14
WX+3.3	31:0	Ref1 Sub-block XY 13
WX+3.2	31:0	Ref0 Sub-block XY 13
WX+3.1	31:0	Ref1 Sub-block XY 12
WX+3.0	31:0	Ref0 Sub-block XY 12

IDM Input Message Phases

Major changes:

- Consists of the 256b source pixel mask.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

DWord	Bits	Name
MX+0.7	31:0	**Source MB Pixel Mask Row 14, 15**
MX+0.6	31:0	**Source MB Pixel Mask Row 12, 13**
MX+0.5	31:0	**Source MB Pixel Mask Row 10, 11**
MX+0.4	31:0	**Source MB Pixel Mask Row 8, 9**
MX+0.3	31:0	**Source MB Pixel Mask Row 6, 7**

DWord	Bits	Name
MX+0.2	31:0	**Source MB Pixel Mask Row 4, 5**
MX+0.1	31:0	**Source MB Pixel Mask Row 2, 3**
MX+0.0	31:16	**Source MB Pixel Mask Row 1**
	15:0	**SourceMB Pixel Mask Row 0** These fields disable a given pixel of the 16x16 source MB. They are arranged from left to right with bit 0 being the left-most pixel and bit 15 being the right-most pixel of the source MB. Format = disable

Return Data Message Phases

Major changes:

Major Change
Many of the fields are not valid output for all message types.
Addition of new message phase, which has the block reference IDs and forward transform skip check data.
Intra chroma distortion and best mode are reported.
All U14 distortion values are now U16.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware ignores these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

DWord	Bits	Description
W0.7	31:16	**Total VME Stalled Clocks:** Counts the number of clocks VME is stalled or starved while processing this request, due to cache misses. Format: U16 ValidMsgType = SIC, IME, FBR
W0.7	15:0	**Total VME Compute Clocks:** Counts the number of clocks VME is processing this request, but not stalled or starved as a result of cache misses. Format: U16 ValidMsgType = SIC, IME, FBR
W0.6	31:26	**Alternate Search Path Length:** Counts the number of unique search units computed by VME for the alternate search path for dual reference or dual search path. If the search path would return to a previously processed SU, it would not be reprocessed and hence not recounted. The value of

DWord	Bits	Description
		[W0.1 15:8] is the overall total search units processed from both paths whereas this value is the contribution only from the second search path. **Note:** Whenever VME is in a mode that processes only a single search path, this field is 0x0. Format: U6, Range of 0-48 ValidMsgType = IME
W0.6	25	**MaxMV Occurred:** This bit is set if the MaxMV event prevented the lowest distortion solution is rejected due to lack of motion vectors. Format: U1 Valid only for Luma Source Size = 16x16. ValidMsgType = IME
W0.6	24	**EarlyIMEStop Occurred:** This bit is set if the EarlyIMEStop threshold is satisfied and IME discontinues searching. Format: U1 ValidMsgType = IME
W0.6	23:16	**Sub-Macroblock Prediction Mode (SubMbPredMode):** If **InterMbMode** is INTER8x8, this field describes the prediction mode of the sub-partitions in the four 8x8 sub-macroblock. It contains four subfields each with 2 bits, corresponding to the four 8x8 sub-macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from **MbType** for a non-BP_8x8 inter macroblock, and carries redundant information as **MbType**. If **InterMbMode** is INTER16x16, INTER16x8 or INTER8x16, this field carries the prediction modes of the sub macroblock (one 16x16, two 16x8, or two 8x16). The unused bits are set to zero. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3] ValidMsgType = SIC, IME, FBR
W0.6	15:8	**Sub-Macroblock Shape (SubMbShape):** This field describes the subdivision of the four 8x8 sub-macroblocks. It contains four subfields each with 2 bits, corresponding to the four 8x8 sub macroblocks in sequential order.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>This field is derived from sub_mb_type for a BP_8x8 or equivalent macroblock.

This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as MbType.

This field is only valid if InterMbMode is INTER8x8; otherwise, it is set to zero.

Bits [1:0]: SubMbShape[0]

Bits [3:2]: SubMbShape[1]

Bits [5:4]: SubMbShape[2]

Bits [7:6]: SubMbShape[3]

ValidMsgType = SIC, IME, FBR</td></tr>
<tr><td>W0.6</td><td>7:0</td><td>Macroblock Intra Structure (MbIntraStruct): This is a bitmask that specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search.

This field is simply copied from the input message (to reduce software overhead of forming the output message to PAK).

<table>
<tr><th>Bits</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>7</td><td>Reserved: MBZ (for IntraPredAvailFlagF – F (pixel[-1,7] available for MbAff)</td></tr>
<tr><td>6</td><td>Reserved: MBZ (for IntraPredAvailFlagA/E – A (left neighbor top half for MbAff)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE/A – A (Left neighbor or Left bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B (Upper neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C (Upper left neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D (Upper right neighbor)</td></tr>
<tr><td>1:0</td><td>ChromaIntraPredMode</td></tr>
</table>

Note: This 8b field is MBZ when IntraComputeType == 1X (when intra is disabled).

ValidMsgType = SIC</td></tr>
<tr><td>W0.5</td><td>31:16</td><td>LumaIntraPredModes[3]</td></tr>
<tr><td>W0.5</td><td>15:0</td><td>LumaIntraPredModes[2]</td></tr>
<tr><td>W0.4</td><td>31:16</td><td>LumaIntraPredModes[1]</td></tr>
</table>

DWord	Bits	Description
W0.4	15:0	**LumaIntraPredModes[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block (Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes. 4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes. **Note:** The LumaIntraPredModes are MBZ when IntraComputeType == 1X (when intra is disabled). ValidMsgType = SIC
W0.3	31:16	**BestChromaIntraDistortion** This field provides the ChromaIntraMode distortion (sum of Cb and Cr dist). Note: this field is MBZ when IntraComputeType == 1X (when intra is disabled). Format = U16 ValidMsgType = SIC
W0.3	15:0	**BestIntraDistortion** This field provides redundant information. The IntraMbMode indicates if this is a 16x16/8x8/4x4 distortion. **Note:** This field is MBZ when IntraComputeType == 1X (when intra is disabled). Format = U16 ValidMsgType = SIC
W0.2	31:16	**SkipRawDistortion** This field contains Skip Raw Distortion which may be used by software to further refine the skip decision. **Note:** this field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
W0.2	15:0	**InterDistortion** This field provides the best inter distortion. Format = U16 ValidMsgType = SIC, IME, FBR
W0.1	31:27	Reserved: MBZ

DWord	Bits	Description
W0.1	26:16	**Sum Ref1 Inter Dist Upper 10 bits (SumInterDistL1Upper)** Contains the sum distortion of all 16x16 Inter shape of Ref1 within the searched SU of this search window. MSB 11 bits only. Format = U11 ValidMsgType = IME
W0.1	15:8	**Search Path Length:** This field returns the number of SU it takes in the integer search. It includes predetermined search path and dynamic search path. Format: U8 ValidMsgType = IME
W0.1	7:4	**Reference 1 border reached:** Bitmask indicating whether any border of reference 1 is reached by one or more motion vectors in the winning inter mode. xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached ValidMsgType = IME
W0.1	3:0	**Reference 0 border reached:** Bitmask indicating whether any border of reference 0 is reached by one or more motion vectors in the winning inter mode. xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached ValidMsgType = IME
W0.0	31	**StreamOutHasNoLocal:** Stream In is the stream out.
W0.0	30	**MinHasLocal:** Part of the results for the minimum distortion contains local results.
W0.0	29	**StreamOutOnlyLocal:** The stream out results don't have any of the stream in results included.
W0.0	28:24	**MvQuantity:** Specifies the number of MVs in packed format (in units of motion vectors). **Note:** This field is provided to help software meet conformance requirements such as maximum number of motion vectors for two consecutive macroblocks.

DWord	Bits	Description
		Format: U5, valid from 0 to 32 ValidMsgType = SIC, IME, FBR
W0.0	23	**ExtendedForm:** This field specifies that **LumaIntraMode**'s are fully replicated in 4x4 sub-blocks respectively. And motion vectors must be in unpacked form as well. This non-DXVA form is used for optimal kernel performance. **This is reserved MBZ and the HW always extends.** ValidMsgType = SIC
W0.0	22:21	Reserved: MBZ
W0.0	20:16	**IntraMbType** This field is encoded to match with the inter type determined as described in the next section. It follows a unified encoding for intra macroblocks according to the AVC Spec. **Note:** This field is MBZ when IntraComputeType == 1X (when intra is disabled). ValidMsgType = SIC
W0.0	15	**Transform8x8Flag (Transform 8x8 Flag)** This field indicates that 8x8 transform is recommended. It is set to 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8. For **IntraMbFlag** = INTER. If **T8x8FlagForInterEn** = 0, this field is set to 0 by VME hardware. If **T8x8FlagForInterEn** = 1, this field is set to 1 if there is no sub macroblock size less than 8x8 (noSubMbPartSizeLessThan8x8Flag = 1). 0: 4x4 integer transform 1: 8x8 integer transform **Note:** This bit is always 0 for non-16x16 source block cases. ValidMsgType = IME, FBR
W0.0	14	**FieldMbFlag** This field indicates the inter prediction result is field or frame. It is always set to **SrcAccess**. 0: Frame macroblock 1: Field macroblock ValidMsgType = SIC, IME, FBR
W0.0	13	Reserved: MBZ
W0.0	12:8	**InterMbType**

DWord	Bits	Description
		This field is encoded to match with the inter type determined as described in the next section. It follows a unified encoding for inter macroblocks according to AVC Spec. ValidMsgType = SIC, IME, FBR
W0.0	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. Unique for AVC standard, within an MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in an MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0. Within a field picture in most coding standard, this field is a constant for the whole field picture. It is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. This field is reserved and MBZ for a progressive frame picture. VME hardware set this field to 1 if the source block is a field block from the bottom field and otherwise sets it to 0. This is accomplished by the following equation using input signals **SrcAccess** and **SrcY**: **SrcAccess** && (bit0(**SrcY**) ==1). 0 = Current macroblock is a field macroblock from the **top** field. 1 = Current macroblock is a field macroblock from the **bottom** field. Equals SrcAccess && SrcFieldPolarity(M1.7[19]) ValidMsgType = SIC, IME, FBR
W0.0	6	Reserved: MBZ
W0.0	5:4	**IntraMbMode** This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if intra mode is enabled. ValidMsgType = SIC
W0.0	3:2	Reserved: MBZ
W0.0	1:0	**InterMbMode** This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if inter mode is enabled. ValidMsgType = SIC, IME, FBR
W1.7 to W1.2	31:0 Each	**MVb[3]** to **MVb[1]**. Motion vectors 3 to 1 for Reference 1, and

DWord	Bits	Description
		MVa[3] to **MVa[1]**. Motion vectors 3 to 1 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W1.1	31:16	**MVb[0].y:** Returning the y-coordinate of Motion Vector 0 for Reference 1, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W1.1	15:0	**MVb[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 1, relative to source MB location. Its meaning is determined by **MbType**. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W1.0	31:16	**MVa[0].y:** Returning the y-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W1.0	15:0	**MVa[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ the first pixel in 6 by 2 block) for Reference 0, relative to source MB location. Its meaning is determined by **MbType**. Hardware Range: [-2048.00 to 2047.75] ValidMsgType = IME **MVa[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 0, relative to source MB location. Its meaning is determined by **MbType**. The returned motion vectors are placed in a fixed data format, with up to 16 motion vectors for one reference and the motion vectors from reference 0 and 1 interleaved. If M1.7:20 (Enable WeightedSAD) is set true (1), then this field and the rest of W1 through W4 are redefined as follows: W1.0 from 31:0 = Minimum 16x16 distortion when applying 1st weighting pattern (16x16_0) W1.1 from 31:0 = Reserved

DWord	Bits	Description
		W1.2 from 31:0 = Minimum 16x16 distortion when applying 2nd weighting pattern (16x16_1) W1.3 from 31:0 = Reserved W1.4 from 31:0 = Minimum 16x16 distortion when applying 3rd weighting pattern (16x16_2) W1.5 from 31:0 = Reserved W1.6 from 31:0 = Minimum 16x16 distortion when applying 4th weighting pattern (16x16_3) W1.7 from 31:0 = Reserved W2 through W4 = Reserved If M0.3:31 (2D 7 by 7 enable) is set true (1), then this field and the rest of W1 through W4 are redefined as follows: W1.0 from 15:0 = returning the x-coordinate of Motion Vector 0 (co-located w/ the first pixel in 6 by 2 block) for Reference 0, relative to source MB location. W1.0 from 31:16 = returning the y-coordinate of Motion Vector 0 (co-located w/ the first pixel in 6 by 2 block) for Reference 0, relative to source MB location. W1.1 from 31:0 = Reserved W1.2 from 31:0 = MVa[1] W1.3 from 31:0 = Reserved W1.4 from 31:0 = MVa[2] W1.5 from 31:0 = Reserved W1.6 from 31:0 = MVa[3] W1.7 from 31:0 = Reserved W2.0 from 31:0 = MVa[4] W2.1 from 31:0 = Reserved W2.2 from 31:0 = MVa[5] W2.3 from 31:0 = Reserved W2.4 from 31:0 = MVa[6] W2.5 from 31:0 = Reserved W2.6 from 31:0 = MVa[7]

DWord	Bits	Description
		W2.7 from 31:0 = Reserved W3.0 from 31:0 = MVa[8] W3.1 from 31:0 = Reserved W3.2 from 31:0 = MVa[9] W3.3 from 31:0 = Reserved W3.4 from 31:0 = MVa[10] W3.5 from 31:0 = Reserved W3.6 from 31:0 = MVa[11] W3.7 from 31:0 = Reserved W4.0 to W4.7 bits 31:0 = Reserved Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W2.7 to W2.0	31:0 Each	**MVb[7]** to **MVb[4]**. Motion vectors 7 to 4 for Reference 1, and **MVa[7]** to **MVa[4]**. Motion vectors 7 to 4 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W3.7 to W3.0	31:0 Each	**MVb[11]** to **MVb[8]**. Motion vectors 11 to 8 for Reference 1, and **MVa[11]** to **MVa[8]**. Motion vectors 11 to 8 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W4.7 to W4.0	31:0 Each	**MVb[15]** to **MVb[12]**. Motion vectors 15 to 12 for Reference 1, and **MVa[15]** to **MVa[12]**. Motion vectors 15 to 12 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W5.7 to W5.1	31:0 Each	**InterDistortion[15] to InterDistortion[2]**. Inter-prediction-distortion associated with motion vector 15 to 2. Its meaning is determined by sub-shape.

DWord	Bits	Description
		See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W5.0	31:16	**InterDistortion[1].** Inter-prediction-distortion with motion vector 1 (co-located with subblock_4x4_1). Its meaning is determined by sub-shape. Format = U16 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W5.0	15:0	**InterDistortion[0].** Inter-prediction-distortion associated with motion vector 0 (co-located with subblock_4x4_0). Its meaning is determined by sub-shape. It must be zero if the corresponding sub-shape is not chosen. This field may be associated with MVa[0] and/or MVb[0], depending on the resulting prediction mode for the sub-block. If the corresponding MV field is created by "duplication", this field must be zero. For 1MVP skip messages, the 16x16 distortion (sad + mv cost + ref cost) is present here. For 4MVP skip messages, the 4 8x8 distortions (sad + mv cost + ref cost) are present here. **Note:** This set of inter-prediction-distortion fields contains detailed information for all sub-shapes. It may be used to assist a multi-pass motion search algorithm. The overall distortion for the macroblock is provided in W0.1. **Format = U16** If M1.7:20 (Enable WeightedSAD) is set true (1), then this field and the rest of W5 are redefined as follows: W5.0 from 15:0 = Minimum 16x16 distortion when applying 1st weighting pattern (16x16_0 with corresponding MV at W1.0) W5.0 from 31:16 = Minimum 16x16 distortion when applying 2nd weighting pattern (16x16_1 with corresponding MV at W1.2) W5.1 from 15:0 = Minimum 16x16 distortion when applying 3rd weighting pattern (16x16_2 with corresponding MV at W1.4) W5.1 from 31:16 = Minimum 16x16 distortion when applying 4th weighting pattern (16x16_3 with corresponding MV at W1.6) W5.7 to W5.2 = Reserved If M0.3:31 (2D 7 by 7 enable) is set true (1), then this field and the rest of W5 is redefined as follows:

DWord	Bits	Description
		W5.0 from 15:0 = **InterDistortion[0]**- Inter-prediction-distortion associated with motion vector 0 (First pixel in 6 by 2 block). W5.0 from 31:16 = **InterDistortion[1]** -Inter-prediction-distortion with motion vector 1 (Second pixel in 6 by 2 block). Its meaning is determined by sub-shape. W5.1 to W5.5 = **InterDistortion[2] to InterDistortion[11]** W5.1 to W5.5 = Reserved ValidMsgType = SIC, IME, FBR
W6.7	31:16	**Max Ref1 Inter Dist (MaxRef1InterDist)** Contains the distortion of the 16x16 Inter shape of Ref1 with the maximum distortion within the searched SU of this search window. Format = U16 ValidMsgType = IME
W6.7	15:0	**Max Ref0 Inter Dist (MaxRef0InterDist)** Contains the distortion of the 16x16 Inter shape of Ref0 with the maximum distortion within the searched SU of this search window. Format = U16 ValidMsgType = IME
W6.6	31:27	Reserved: MBZ
W6.6	26:0	**Sum Ref0 Inter Dist (SumRef0InterDist)** Contains the sum distortion of all 16x16 Inter shape of Ref0 within the searched SU of this search window. Format = U27 ValidMsgType = IME
W6.5	31:16	**Block 0 Chroma Cr Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
W6.5	15:0	**Block 0 Chroma Cb Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold.

DWord	Bits	Description
		Note: This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
W6.4	31:16	**Sum Ref1 Inter Dist lower 16 bits (SumInterDistL1Lower)** Contains the sum distortion of all 16x16 Inter shape of Ref1 within the searched SU of this search window. Lower 16 bits. Format = U16 ValidMsgType = IME
W6.4	15:8	**Block 0 Chroma Cr NZC** Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
W6.4	7:0	**Block 0 Chroma Cb NZC** Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
W6.3	31:16	**Block 3 Luma Coeff Magnitude Clip Sum**
W6.3	15:0	**Block 2 Luma Coeff Magnitude Clip Sum**
W6.2	31:16	**Block 1 Luma Coeff Magnitude Clip Sum**
W6.2	15:0	**Block 0 Luma Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
W6.1	31:24	**Block 3 Luma NZC**

DWord	Bits	Description
W6.1	23:16	**Block 2 Luma NZC**
W6.1	15:8	**Block 1 Luma NZC**
W6.1	7:0	**Block 0 Luma NZC** Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
W6.0	31:28	**Bwd Block 3 RefID**
W6.0	27:24	**Fwd Block 3 RefID**
W6.0	23:20	**Bwd Block 2 RefID**
W6.0	19:16	**Fwd Block 2 RefID**
W6.0	15:12	**Bwd Block 1 RefID**
W6.0	11:8	**Fwd Block 1 RefID**
W6.0	7:4	**Bwd Block 0 RefID** Reference ID for backward block 0. **Note:** Even if shape is 16x16, this field is defined per block, hence VME will replicate the RefID for larger shapes. Replication happens only in IME. For CRE (SIC/FBR), this is a pass through field. Format = U4 ValidMsgType = SIC, IME, FBR
W6.0	3:0	**Fwd Block 0 RefID** Reference ID for forward block 0. **Note:** Even if shape is 16x16, this field is defined per block, hence VME will replicate the RefID for larger shapes. Replication happens only in IME. For CRE (SIC/FBR), this is a pass through field.

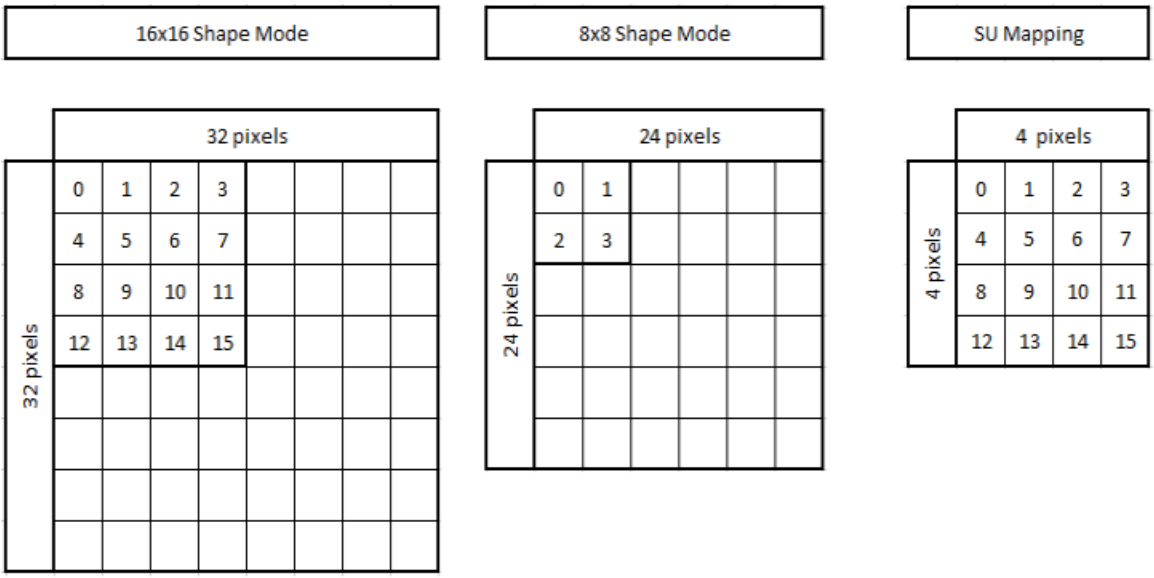
DWord	Bits	Description
		Format = U4 ValidMsgType = SIC, IME, FBR

IME StreamOut

IME Streamout follows the same format as the IME Streamin message phases (IME2-IME5).

IDM Stream-Out

Because the IDM output message only contains distortion values without their corresponding motion vectors, a direct mapping system is used to derive the MV associated with a given distortion.



This mapping allows the user to derive the location of a given distortion easily by applying the following equations.

ShapeMode == 16x16	ShapeMode == 8x8
MV.x = (GRF# % 4) * 4 + W# % 4 + Ref.x	**MV.x = ((GRF# / 4) % 2) * 4 + W# % 4 + Ref.x**
MV.y = (GRF# / 4) * 4 + W# / 4 + Ref.y	**MV.y = ((GRF# / 4) / 2) * 4 + W# / 4 + Ref.y**

- GRF#: The 256b return message phase number.
- W#: The 16 16b word within each 256b return message.

IDM16x16 Streamout Message Format

DWord	Bits	Description
W15	31:0	**Distortion Mesh Block 15 Search Points 0-15**
W14	31:0	**Distortion Mesh Block 14 Search Points 0-15**
W13	31:0	**Distortion Mesh Block 13 Search Points 0-15**
W12	31:0	**Distortion Mesh Block 12 Search Points 0-15**
W11	31:0	**Distortion Mesh Block 11 Search Points 0-15**
W10	31:0	**Distortion Mesh Block 10 Search Points 0-15**
W9	31:0	**Distortion Mesh Block 9 Search Points 0-15**
W8	31:0	**Distortion Mesh Block 8 Search Points 0-15**
W7	31:0	**Distortion Mesh Block 7 Search Points 0-15**
W6	31:0	**Distortion Mesh Block 6 Search Points 0-15**
W5	31:0	**Distortion Mesh Block 5 Search Points 0-15**
W4	31:0	**Distortion Mesh Block 4 Search Points 0-15**
W3	31:0	**Distortion Mesh Block 3 Search Points 0-15**
W2	31:0	**Distortion Mesh Block 2 Search Points 0-15**
W1	31:0	**Distortion Mesh Block 1 Search Points 0-15**
W0.7	31:16	**Distortion Mesh Block 0 Search Point 15**
	15:0	**Distortion Mesh Block 0 Search Point 14**
W0.6	31:16	**Distortion Mesh Block 0 Search Point 13**
	15:0	**Distortion Mesh Block 0 Search Point 12**
W0.5	31:16	**Distortion Mesh Block 0 Search Point 11**

DWord	Bits	Description
	15:0	**Distortion Mesh Block 0 Search Point 10**
W0.4	31:16	**Distortion Mesh Block 0 Search Point 9**
	15:0	**Distortion Mesh Block 0 Search Point 8**
W0.3	31:16	**Distortion Mesh Block 0 Search Point 7**
	15:0	**Distortion Mesh Block 0 Search Point 6**
W0.2	31:16	**Distortion Mesh Block 0 Search Point 5**
	15:0	**Distortion Mesh Block 0 Search Point 4**
W0.1	31:16	**Distortion Mesh Block 0 Search Point 3**
	15:0	**Distortion Mesh Block 0 Search Point 2**
W0.0	31:16	**Distortion Mesh Block 0 Search Point 1**
	15:0	**Distortion Mesh Block 0 Search Point 0** This is the distortion value at the location (0,0) within the search window. Each 256b (8 DW) message phase contains 16 distortion values for a given 4x4 search unit of the search window. Each word of a message phase contains a single 16x16 distortion at a given search point. The individual distortion values are arranged in raster-scan (row-major order) within the 4x4. For ShapeMode == 16, the 16 message phases comprise the 16 SUs (4x4) of the 32x32 search window for each 16x16 distortion and are returned in raster-scan (row-major order). Format = U16

IDM8x8 Streamout Message Format

DWord	Bits	Description
W15	31:0	**Distortion Mesh Block 3 SU3 Search Points 0-15**
W14	31:0	**Distortion Mesh Block 2 SU3 Search Points 0-15**
W13	31:0	**Distortion Mesh Block 1 SU3 Search Points 0-15**
W12	31:0	**Distortion Mesh Block 0 SU3 Search Points 0-15**
W11	31:0	**Distortion Mesh Block 3 SU2 Search Points 0-15**

DWord	Bits	Description
W10	31:0	**Distortion Mesh Block 2 SU2 Search Points 0-15**
W9	31:0	**Distortion Mesh Block 1 SU2 Search Points 0-15**
W8	31:0	**Distortion Mesh Block 0 SU2 Search Points 0-15**
W7	31:0	**Distortion Mesh Block 3 SU1 Search Points 0-15**
W6	31:0	**Distortion Mesh Block 2 SU1 Search Points 0-15**
W5	31:0	**Distortion Mesh Block 1 SU1 Search Points 0-15**
W4	31:0	**Distortion Mesh Block 0 SU1 Search Points 0-15**
W3	31:0	**Distortion Mesh Block 3 SU0 Search Points 0-15**
W2	31:0	**Distortion Mesh Block 2 SU0 Search Points 0-15**
W1	31:0	**Distortion Mesh Block 1 SU0 Search Points 0-15**
W0.7	31:16	**Distortion Mesh Block 0 SU0 Search Point 15**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 14**
W0.6	31:16	**Distortion Mesh Block 0 SU0 Search Point 13**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 12**
W0.5	31:16	**Distortion Mesh Block 0 SU0 Search Point 11**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 10**
W0.4	31:16	**Distortion Mesh Block 0 SU0 Search Point 9**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 8**
W0.3	31:16	**Distortion Mesh Block 0 SU0 Search Point 7**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 6**
W0.2	31:16	**Distortion Mesh Block 0 SU0 Search Point 5**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 4**
W0.1	31:16	**Distortion Mesh Block 0 SU0 Search Point 3**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 2**
W0.0	31:16	**Distortion Mesh Block 0 SU0 Search Point 1**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 0** This is the distortion value at the location (0,0) within the search window. Each 256b (8 DW) message phase contains 16 distortion values for a given 4x4 search unit of the search window. Each word of a message phase contains a single 8x8 distortion at a given search point. The individual distortion values are arranged in raster-scan (row-major order) within the 4x4. For ShapeMode == 8, the first 4 message phases comprise the 1st SU of the 24x24 search window for all 4 block8x8_0-block8x8_1 distortions and are returned in raster-scan (row-major order). The next 4 message phases contain the 2nd SU and so on. Format = U16

Sample_8x8 State

This section contains different state definitions.

This state definition is used only by the *deinterlace* message. This state is stored as an array of up to 8 elements, each of which contains the dwords described here. The start of each element is spaced 8 dwords apart. The first element of the array is aligned to a 32-byte boundary. The index with range 0-7 that selects which element is being used is multiplied by 2 to determine the **Sampler Index** in the message descriptor.

SURFACE_STATE for Deinterlace, sample_8x8, and VME

This section contains media surface state definitions.

MEDIA_SURFACE_STATE

Restrictions: The Faulting modes described in the MEMORY_OBJECT_CONTROL_STATE should be set to the same for the multi-surface Video Analytics functions like “LBP Correlation” and “Correlation Search” for both the surfaces.

SAMPLER_STATE for Sample_8x8 Message

SAMPLER_STATE has different formats, depending on the message type used. For SNB+, the sample_8x8 and deinterlace messages use a different format of SAMPLER_STATE as detailed in the corresponding sections.

Sampler State
SAMPLER_STATE

- IEF Filter Type was dropped and is assumed to be Detailed filter
- IEF Filter Size was dropped and assumed to be 5x5.
- IEF Bypass – If we have Y/G-channel masked then the IEF bypass should always be forced to 1.
- Dword 0-15 is valid only when the function is “AVS Scaling”
- When IEF is Bypass, IEF ignores this state and need not store the state. AVS scaling is still on and uses this state. This would need to be indicated by the tag (maybe) returned so that AVS does not send the writes to IEF.
- Function Convolve : State is 144DWs
- Function MinMaxfilter/Erode/Dilate : State is 8DWs
- Function MinMax/BoolCentroid/Centroid : State is 0DWs. The sampler state is not required.
-

  This state definition is used only by the *sample_8x8* message only for specific function and the length of the state varies according to the function programmed in the message header.

For AVS and Convolve the state is stored as an array of up to 48 elements (**192** DWs), (upsize to 64 elements, 256DWs), each of which contains the dwords described here. The start of each of this state is spaced **256** dwords apart. The first element of the array is aligned to a 64-byte boundary. The **sampler index** in the message descriptor is multiplied by **32** to determine the offset from the base where the sampler state is to be read from. Sampler states with lower foot print than 32 elements should be packed at lower offsets and this sampler state for sample_8x8 message should be kept at the end. We can reuse existing sampler_index if the result of the multiplication of 32 is not overlapping with the existing states already programmed at the lower offsets. Two adjacent state of this type should have a space of 2 sample index.

For MinMaxFilter, Erode and Dilate the state is stored as an array of up to 2 elements (8 DWs), each of which contains the dwords as described in following section. The start of each of this state is spaced **8** dwords apart. The first element of the array is aligned to a 32-byte boundary. The **sampler index** in the message descriptor is multiplied by **2** to determine the offset from the base where the sampler state is to be read from. Sampler states with lower foot print than 2 elements should be packed at lower offsets and this sampler state for sample_8x8 message should be kept after it. Sampler states with larger footprint (192DWs) as described earlier should be packed after this. We can reuse existing sampler_index if the result of the multiplication of 2 is not overlapping with the existing states already programmed at the lower offsets. Two adjacent state of this type should have a space of 1 sample index.

Sampler State
SAMPLER_STATE_8x8_AVS
SAMPLER_STATE_8x8_AVS_COEFFICIENTS

Most of the existing functions are carried over and the states are programmed in exactly the same way. The following functions are added in SKL and use sampler state:

- Convolve (00000) & SKL_mode: 512 DWords or 128 Elements.

  Note: Convolve & BDW_mode is the same as for BDW with no change.
- 1D Vertical / Horizontal Convolution (function 01000 and 01001): 32 DWords or 8 Elements.
- 1 Pixel Convolution (Same as Convolve & BDW_mode): 144 DWords.
- AVS Scaling: 280 DWords or 70 elements. Occupies 128 Elements for each state.

From the above list we can see that we are required to support 128 Elements and 8 Elements which is new in SKL and the rest of element size are already supported in BDW. For this case the sampler_index is multiplied by 128 (for "2D Convolution and SKL_Mode" and "AVS Scaling") and 8 (for 1D convolution) respectively. Lower state sizes should use lower index and higher states sizes should use higher index and should be aligned to the sampler state size as in BDW.

Description
Function FloodFill/LBPCreation/CorrelationSearch: State is 0 DWords.
Function 1PixelConvolution: State is 192 DWords.
Function 1D Vertical/1D Horizontal Convolution: State is 32 DWords.
Function Convolution & SKL_mode: State is 512 DWords.
Function AVS Scaling: State is 280 DWords.

SIMD32/64 Messages

Initiating Message

SIMD32/64 Payload

SIMD32 Payload

DWord	Bits	Description
M1.7	31:0	**Reserved**
M1.6	31:0	**U 2nd Derivative** Defines the change in the delta U for adjacent pixels in the X direction. Format = IEEE_Float in normalized space.
M1.5	31:0	**Delta V**: defines the difference in V for adjacent pixels in the Y direction. **Programming Notes:** • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • This field is ignored for the deinterlace message type. • Negative **Delta V** are not supported and should be clamped to 0. Format = IEEE_Float in normalized space.
M1.4	31:0	**Delta U:** defines the difference in U for adjacent pixels in the X direction. **Programming Notes:** • **Delta U** multiplied by **Width** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • This field is ignored for the deinterlace message type. • Negative **Delta U** are not supported and should be clamped to 0. Format = IEEE_Float in normalized space.

DWord	Bits	Description
M1.3	31:0	**Pixel 0 V Address** Format: sample_unorm* : IEEE_Float in normalized space. Deinterlace: U32 (Range: [0,2046]) **:** Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
M1.2	31:0	**Pixel 0 U Address** Format: sample_unorm* : IEEE_Float in normalized space. Deinterlace: U32 (Range: [0,4095]) Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
M1.1	31:0	**Reserved**
M1.0	31:0	Ignored

SIMD64 Payload

IEF bypass has been moved from state to the payload. This needs to be forced to zero for all other functionality other than AVS. IEF bypass would not be controlled through inline parameter to the kernel for AVS with IEF off case. For Planar surfaces, the IEF bypass needs to be forced to zero for U,V channels by the kernel.

DWord	Bits	Description
M1.7	31:28	**Functionality** Bit 22 FunctionMode = 0:

Value	Function
0000b	AVS scaling
0001b	Convolve
0010b	MINMAX
0011b	MINMAXFILTER
0100b	ERODE
0101b	Dilate
0110b	BoolCentroid / BoolSum
0111b	Centroid
1000b	1D Vertical Convolution

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>1001b</td><td>1D Horizontal Convolution</td></tr>
<tr><td>1010b</td><td>1 Pixel Convolution</td></tr>
<tr><td>1011b</td><td>Flood Fill</td></tr>
<tr><td>1100b</td><td>LBP Creation</td></tr>
<tr><td>1101b</td><td>LBP Correlation</td></tr>
<tr><td>1110b</td><td>Reserved</td></tr>
<tr><td>1111b</td><td>Correlation Search</td></tr>
</table>
</td></tr>
<tr><td>M1.7</td><td>27</td><td>IEF Bypass
Ignored for all functions as it will always be IEF bypass except for AVS scaling operation.</td></tr>
<tr><td>M1.7</td><td>26:23</td><td>Control
[26:25] Message_Seq:

When Functionality is AVS:
11: 4x4 (only supported with Shuffle_OutputWriteback=1)
10: 16x8
01: 8x4 (only supported with Shuffle_OutputWriteback=1)
00: 16x4

When Functionality is Convolve/MinMaxFilter/LBP Correlation/LBP Creation:
11: 1x1 – This would be used in MinMaxFilter case & 1pixel convolve only for operation in scattered regions and don’t want to waste the power. This mode is not used and is reserved for 2D/1D Convolve, LBP Creation, and LBP Correlation.
10: 16x1 – This would be used in case we require the operation in scattered regions and don’t want to waste the throughput. 1-D Horizontal this will be 1x16. Not supported in LBP creation and LBP correlation.
0x: 16x4 – except for 1-D Horizontal this will be 4x16; illegal for 1 pixel convolve.

When Functionality is Erode/Dilate:
11: 32x1 – This would be used in case we require the operation in scattered regions and don’t want to waste the throughput.
10: 64x1 – This would be used in case we require the operation in scattered regions and don’t want to waste the throughput.</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>01: 32x4
00: 64x4
When Functionality is MINMAX/BoolCentroid/Centroid
1x: These modes should always force this to 1 since we don’t have SI sequencing.
0x: Unexpected behavior

When Functionality is FloodFill:
[26]: Reserved
[25]:
1 - 8-connected is used for floodfill.
0 - 4-connected is used for floodfill.

[24:23]:
When Functionality is MinMaxFilter/MinMax/LBP Creation Only:
11: Reserved
10: Disable Max Filter. (Min Filter only)/5x5 LBP
01: Disable Min Filter. (Max Filter only)/3x3 LBP
00: Both Min and Max filter/Both 3x3 and 5x5 LBP

[24:23]:
When Functionality is MinMaxFilter/LBP Creation Only:
11: Reserved
10: Disable Max Filter. (Min Filter only)/5x5 LBP
01: Disable Min Filter. (Max Filter only)/3x3 LBP
00: Both Min and Max filter/Both 3x3 and 5x5 LBP

When Functionality is Convolve:
<table><tr><th>Description</th></tr><tr><td>1x: Reserved</td></tr></table>Reserved for all other functionality.</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>M1.7</td><td>22</td><td>Reserved: MBZ</td></tr>
<tr><td>M1.7</td><td>21:0</td><td>Group ID Number

Used to group messages for reorder for sample_8x8 messages. All messages with the same Group ID must have the following in common: SURFACE_STATE, SAMPLER_STATE, Destination register on send instruction, M0, and M1 except for Horizontal and Vertical Block Number.</td></tr>
<tr><td>M1.6</td><td>31:0</td><td>Function = AVS:

U 2nd Derivative: Defines the change in the delta U for adjacent pixels in the X direction.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>SIMD64 Payload</td></tr><tr><td colspan="2">This field is ignored for messages other than sample_8x8.
(64 – (2*du))/35 >= ddu >= -du/18
2*ddu <= du</td></tr></table>
Format = IEEE_Float in normalized space

Function = FloodFill:

[31:16] Pixel mask of the bottom adjacent pixels in horizontal direction (Row 8 of 16x8 pixel).

[15:0] Pixel mask of the top adjacent pixels in horizontal direction (Row -1 of 16x8 pixel).

If function is not AVS or Floodfill, this field is ignored and assumed to be 0 (Reserved: MBZ).</td></tr>
<tr><td>M1.5</td><td>31:0</td><td>Function = AVS:

Delta V: defines the difference in V for adjacent pixels in the Y direction.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>SIM64 O</td></tr><tr><td colspan="2">The normalized value should be less than 1.
Delta V multiplied by Height in SURFACE_STATE must be less than 16 for the sample_8x8 message type.
This field is ignored for the deinterlace message type and VA sample_8x8 messages.
Negative Delta V is not supported and should be clamped to 0.</td></tr></table></td></tr>
</table>

DWord	Bits	Description
		Function = Correlation Search: [31:0] Normalized vertical origin of the reference image.%2

Vertical Block Number Restrictions

	Function	Msg Seq	Vblk Multiplier	Restrictions / Comments
Dword	M1.7	M1.7		Vertical block (vblk) is used for vertical offset except in 1D horizontal convolution where it is used as horizontal offset.
Bits	[31:28]	[26:25]		
	AVS Scaling	16x4	4	With Rotation of 90 and 270 the vblk should not be used and should be zero always.
		16x8	8	
		8x4		
		4x4	4	Vblk is supported with Rotation.
	Convolve	16x4	4	
		16x1	1	
	MinMax		0	Vblk is not used and should be zero.
	MinMaxFilter	16x4	4	
		16x1 or 1x1	1	
	Erode or Dilate	32x1 or 64x1	1	
		32x4 or 64x4	4	
	BoolCentroid		0	Vblk is not used and should be zero.
	Centroid		0	Vblk is not used and should be zero.
	1D Vertical Convolution	16x4	4	
		16x1	1	
	1D Horizontal Convolution	4x16	4	Vblk is used for horizontal offset rather than vertical in this function.
		1x16	1	
	1 Pixel Convolution	16x1, 1x1	0	Vblk is not used and should be zero.
	FloodFill	-	0	Vblk is not used and should be zero.
	LBP Creation	16x4	4	
	LBP Correlation	16x4	0	
	Correlation Search	-	0	Vblk is not used and should be zero.

Payload Parameter Definition

The table below shows all of the message types supported by the sampling engine. The **Message Type** field in the message descriptor determines which message is being sent. The **SIMD Mode** field determines the number of instances (i.e. pixels) and the formatting of the initiating and writeback messages. The **Header Present** field determines whether a header is delivered as the first phase of the message or the default header from R0 of the thread's dispatch is used. The **Message Length** field is used to vary the number of parameters sent with each message. Higher-numbered parameters are optional, and default to a value of 0 if not sent but needed for the surface being sampled.

The message lengths are computed as follows, where "N" is the number of parameters ("N" is rounded up to the next multiple of 4 for SIMD4x2), and "H" is 1 if the header is present, 0 otherwise. The maximum message length allowed to the sampler is 11.

SIMD Mode	Message Length
SIMD4x2	H + (N/4)
SIMD8 SIMD8D	H + N
SIMD16	H + (2*N)

The response lengths are computed as follows:

SIMD Mode		Response Length Return Format = 32-bit	Response Length Return Format = 16-bit ***
SIMD4x2		1	not allowed
SIMD8	sample+killpix	5	not allowed
	all other message types	4	2 **
SIMD16		8 *	4 *

* For SIMD16, phases in the response length are reduced by 2 for each channel that is masked.

** : For SIMD8*, phases in the response length are reduced by 1 for each channel that is masked.

*** only

SIMD16 messages with six or more parameters exceed the maximum message length allowed, in which case they are not supported. This includes some forms of sample_b_c, sample_l_c, and gather4_po_c message types. Note that even for these messages, if 5 or fewer parameters are included in the message, the SIMD16 form of the message is allowed. SIMD16 forms of sample_d and sample_d_c are not allowed, regardless of the number of parameters sent.

: If **Header Present** is *disabled*, **Response Length** can be set to values in the table below for SIMD8 and SIMD16 messages other than *sample+killpix*. The setting of **Response Length** determines the **Write Channel Mask** fields that are used by the hardware according to the table below. **Response Length** values not indicated in the table are not valid.

SIMD Mode	Response Length	Alpha Write Channel Mask	Blue Write Channel Mask	Green Write Channel Mask	Red Write Channel Mask

SIMD Mode	Response Length	Alpha Write Channel Mask	Blue Write Channel Mask	Green Write Channel Mask	Red Write Channel Mask
SIMD8*	1	1	1	1	0
	2	1	1	0	0
	3	1	0	0	0
	4	0	0	0	0
SIMD16*	2	1	1	1	0
	4	1	1	0	0
	6	1	0	0	0
	8	0	0	0	0

: If **Pixel Fault Mask Enable** is enabled, response length must be set to a value one larger than those indicated in the tables above.

: Response Length of zero is allowed on all SIMD8* and SIMD16* sampler messages except sample+killpix, resinfo, sampleinfo, LOD, and gather4*. Header Present must be enabled and Pixel Null Mask Enable must be disabled. The Write Channel Mask for all four channels in the header must be 0 (channel enabled). A shader containing one or more of these messages is not allowed to send any render target read or write messages to the data port. When response length is set to zero, the following behavior occurs:

- When the sampler completes processing of the message, the resulting channels are delivered as input to the Render Cache Data Port in the form of a Render Target Write message without a header. Refer to the Data Port section for more details on this message.
- The fields normally set in the message descriptor for Render Target Write are set as follows:
  - End of Thread (EOT) comes from the EOT on the sampler message. These response length zero sampler messages are the only sampler messages allowed to have EOT enabled.
  - Last Render Target Select is set to true.
  - Slot Group Select is derived from EOT: If EOT is enabled, set to SLOTGRP_LO, otherwise SLOTGRP_HI.
  - RT Write Message Type is derived from the sampler's SIMD Mode field as follows: SIMD8 => SIMD8_LO, SIMD16 => SIMD16
  - Binding Table Index is set to the value set in the **Render Target Binding Table Index** field of the message header.

SIMD8D Messages (only):

Message Type	Mnemonic	Parameters										
		0	1	2	3	4	5	6	7	8	9	10
00000	sample	u0	u1	v0	v1	r	mlod					
00001	sample_b	bias	u0	u1	v0	v1	r	mlod				
00010	sample_l	lod	u0	u1	v0	v1	r					

Message Type	Mnemonic	Parameters										
		0	1	2	3	4	5	6	7	8	9	10
00011	sample_c	ref	u0	u1	v0	v1	r	mlod				
00100	sample_d	u0	u1	dudx	dudy	v0	v1	dvdx	dvdy	r	mlod	
00101	sample_b_c	ref	bias	u0	u1	v0	v1	r				
00110	sample_l_c	ref	lod	u0	u1	v0	v1	r				
01000	gather4	u0	u1	v0	v1	r						
01001	LOD	u0	u1	v0	v1							
10000	gather4_c	ref	u0	u1	v0	v1	r					
10001	gather4_po	u0	u1	v0	v1	offu	offv	r				
10010	gather4_po_c	ref	u0	u1	v0	v1	offu	offv	r			
10100	sample_d_c	ref	u0	u1	dudx	dudy	v0	v1	dvdx	dvdy	r	
10110	sample_min	u0	u1	v0	v1							
10111	sample_max	u0	u1	v0	v1							
11000	sample_lz	u0	u1	v0	v1	r						
11001	sample_c_lz	ref	u0	u1	v0	v1	r					

SIMD4x2 Messages

Message Type	Mnemonic	Parameters										
		0	1	2	3	4	5	6	7	8	9	10
00010	sample_l	u	v	r	ai	lod						
00100	sample_d	u	v	r	ai	dudx	dudy	dvdx	dvdy	drdx	drdy	mlod *
00110	sample_l_c	u	v	r	ai	ref	lod					
00111	ld	u	v	r	lod							
01000	gather4	u	v	r	ai							
01010	resinfo	lod										
01011	sampleinfo											
10000	gather4_c	u	v	r	ai	ref						
10001	gather4_po	u	v	r	ai	offu	offv					
10010	gather4_po_c	u	v	r	ref	offu	offv					
10100	sample_d_c	u	v	r	ai	dudx	dudy	dvdx	dvdy	drdx	drdy	ref
11100	ld2dms_w	u	v	r	lod *	si	mcsl	mcsh				
11101	ld_mcs	u	v	r	lod *							
11110	ld2dms	u	v	r	lod *	si	mcs					

* These parameters are allowed only for .

SIMD32/SIMD64 Messages

Message Type	mnemonic	Payload Layout	Message Length	Response Length
00000	sample_unorm	Pixel Shader	H + 1	8 **
00010	sample_unorm+killpix	Pixel Shader	H + 1	9 **
01000	deinterlace	Pixel Shader	H + 1	†
01100	sample_unorm	Media	H + 1	8 **
01010	sample_unorm+killpix	Media	H + 1	9 **
01011	sample_8x8	Media	H + 1	16 *
11111	cache_flush	no payload	1	1

* These parameters are allowed only for .

SIMD32/SIMD64 Messages

Message Type	mnemonic	Payload Layout	Message Length	Response Length
00000	sample_unorm	Pixel Shader	H + 1	8 **
00010	sample_unorm+killpix	Pixel Shader	H + 1	9 **
01000	deinterlace	Pixel Shader	H + 1	†
01100	sample_unorm	Media	H + 1	8 **
01010	sample_unorm+killpix	Media	H + 1	9 **
01011	sample_8x8	Media	H + 1	16 *
11111	cache_flush	no payload	1	1

* For sample_8x8, phases in the response length are reduced by 4 for each channel that is masked.

** For sample_unorm, phases in the response length are reduced by 2 for each channel that is masked.

† For deinterlace, response length depending on certain state fields. Refer to writeback message definition for details.

SIMD32_64 Message Descriptor

Please refer to the 3D Sampler Message Descriptor definition at **Message Descriptor - Sampling Engine** .

SIMD32_64 Message Header

Please refer to the 3D Sampler Message Header definition at Message Header.

Message Header

The message header for the sampling engine is the same regardless of the message type. If the header is not present, the behavior is as if the message was sent with all fields in the header set to zero (write channel masks are all enabled and offsets are zero). When Response length is 0 for sample_8x8 message

then the data from sampler is directly written out to memory using media write message. Message header needs to be present if mid-thread pre-emption is required.

DWord	Bits	Description
M0.5	31:5	Ignored
M0.5	4:0	**Output Format** Only for Sample_8x8 message with direct HDC write. 0 YCRCB_NORMAL 1 YCRCB_SWAPUVY 2 YCRCB_SWAPUV 3 YCRCB_SWAPY 4 PLANAR_420_8 (NV12 only) 5 PLANAR_Y8_UNORM 6 PLANAR_Y16_SNORM 7 Reserved 8 Reserved 9 Reserved 10 Reserved 11 Reserved 12 Reserved 13 Reserved 14 Reserved 15 Reserved 16 Reserved 17 PLANAR_Y32_UNORM 18 Reserved
M0.4	31:16	Destination Y Address (u16)
M0.4	15:0	Destination X Address in bytes (u16) Note that X Address should always be DWord-aligned.
M0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the sampler state table. This field is ignored for "ld" and "resinfo" message types. This pointer is relative to the **Dynamic State Base Address**. Format = DynamicStateOfffset[31:5]
M0.3	4:1	Ignored
M0.3	0	Ignored

DWord	Bits	Description
M0.2	31:24	**Render Target or Destination Binding Table Index:** Specifies the index into the binding table for the render target or HDC for messages with response length of zero (the binding table index for the sampler surface is in the message descriptor). Format = U8 Range = [0,255]
M0.2	23	**Pixel Null Mask Enable:** Specifies whether the writeback message includes an extra phase indicating the pixel null mask. Refer to the "Writeback Message" section for details on format. This field must be disabled for sample+killpix and for all SIMD32/64 messages. Format = Enable Ignored for Sample_8x8 message
M0.2	22	**SIMD Mode Extension:** If SIMD Mode in the message descriptor is set to SIMD8D / SIMD4x2, this field specifies which mode is used. For other SIMD Modes, this field is ignored. 0: SIMD8D 1: SIMD4x2
M0.2	21	Ignored
M0.2	20	Ignored
M0.2	19:18	**SIMD32/64 Output Format Control** Specifies the output format of SIMD32/64 messages (sample_unorm* and sample_8x8). Ignored for other message types. Refer to the writeback message formats for details on how this field affects returned data. 0: 16 bit Full 1: 16 bit Chrominance Downsampled 2: 8 bit Full 3: 8 bit Chrominance Downsampled **Programming Note** **Context:** Message Header This field is ignored for sample_8x8 messages if the Function is not AVS and MinMaxFilter. For MinMaxFilter only 16bit Full and 8bit Full modes are supported. **Programming Note**

DWord	Bits	Description
		Context: Message Header This field is not used for HDC write messages.
M0.2	17:16	**Gather4 Source Channel Select:** Selects the source channel to be sampled in the gather4* messages. Ignored for other message types. 0: Red channel 1: Green channel 2: Blue channel 3: Alpha channel Note that for gather4*_c messages, this field must be set to 0 (Red channel).
M0.2	15	**Alpha Write Channel Mask:** Enables the alpha channel to be written back to the originating thread. 0: Alpha channel is written back. 1: Alpha channel is not written back. **Programming Note** **Context:** Message Header • A message with all four channels masked is not allowed. • This field is ignored for the deinterlace message. • This field must be set to zero for sample_8x8 in VSA mode. • This field must be set to zero for all gather4* messages. **Programming Note** **Context:** Message Header For Sample_8x8 messages, Alpha/Blue/Red channels should be always masked (set to 1) and only Green channel is enabled (set to 0).
M0.2	14	**Blue Write Channel Mask:** See Alpha Write Channel Mask.
M0.2	13	**Green Write Channel Mask:** See Alpha Write Channel Mask.
M0.2	12	**Red Write Channel Mask:** See Alpha Write Channel Mask.
M0.2	11:8	**U Offset:** The u offset from the _aoffimmi modifier on the “sample” or “ld” instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement. **Programming Note**

DWord	Bits	Description
		Context: Message Header • This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages. • This field is ignored if the "offu" parameter is included in the gather4* messages.
M0.2	7:4	**V Offset:** The v offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note** **Context:** Message Header • This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages. • This field is ignored if the "offu" parameter is included in the gather4* messages.
M0.2	3:0	**R Offset:** The r offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
M0.1	31:0	Ignored
M0.0	31:0	Ignored

SIMD32_64 Payload Parameter Definition

Please refer to the 3D Sampler Payload Parameter Definition at Payload Parameter Definition.

SIMD32_64 Message Types

Please refer to the 3D Sampler Message Types definition at Message Types.

Writeback Message

SIMD32

Sample_unorm*

Pixels are numbered as follows:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3 (using 16 bit Full mode as an example).

"16 bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 11 & 10 Red**
W0.4		**Pixel 9 & 8 Red**
W0.3		**Pixel 7 & 6 Red**
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 31 & 30 Red**
W1.6		**Pixel 29 & 28 Red**

DWord	Bit	Description
W1.5		**Pixel 27 & 26 Red**
W1.4		**Pixel 25 & 24 Red**
W1.3		**Pixel 23 & 22 Red**
W1.2		**Pixel 21 & 20 Red**
W1.1		**Pixel 19 & 18 Red**
W1.0		**Pixel 17 & 16 Red**
W2		**Pixels 15:0 Green**
W3		**Pixels 31:16 Green**
W4		**Pixels 15:0 Blue**
W5		**Pixels 31:16 Blue**
W6		**Pixels 15:0 Alpha**
W7		**Pixels 31:16 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bit	Description
W0.7	31:16	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 28 Red**
W0.6		**Pixel 26 & 24 Red**
W0.5		**Pixel 22 & 20 Red**

DWord	Bit	Description
W0.4		**Pixel 18 & 16 Red**
W0.3		**Pixel 14 & 12 Red**
W0.2		**Pixel 10 & 8 Red**
W0.1		**Pixel 6 & 4 Red**
W0.0		**Pixel 2 & 0 Red**
W1.7	31:16	**Pixel 15 Green**
	15:0	**Pixel 14 Green**
W1.6		**Pixel 13 & 12 Green**
W1.5		**Pixel 11 & 10 Green**
W1.4		**Pixel 9 & 8 Green**
W1.3		**Pixel 7 & 6 Green**
W1.2		**Pixel 5 & 4 Green**
W1.1		**Pixel 3 & 2 Green**
W1.0		**Pixel 1 & 0 Green**
W2.7		**Pixel 31 & 30 Green**
W2.6		**Pixel 29 & 28 Green**
W2.5		**Pixel 27 & 26 Green**
W2.4		**Pixel 25 & 24 Green**
W2.3		**Pixel 23 & 22 Green**
W2.2		**Pixel 21 & 20 Green**

DWord	Bit	Description
W2.1		**Pixel 19 & 18 Green**
W2.0		**Pixel 17 & 16 Green**
W3.7	31:16	**Pixel 30 Blue**
	15:0	**Pixel 28 Blue**
W3.6		**Pixel 26 & 24 Blue**
W3.5		**Pixel 22 & 20 Blue**
W3.4		**Pixel 18 & 16 Blue**
W3.3		**Pixel 14 & 12 Blue**
W3.2		**Pixel 10 & 8 Blue**
W3.1		**Pixel 6 & 4 Blue**
W3.0		**Pixel 2 & 0 Blue**
W4.7	31:16	**Pixel 15 Alpha**
	15:0	**Pixel 14 Alpha**
W4.6		**Pixel 13 & 12 Alpha**
W4.5		**Pixel 11 & 10 Alpha**
W4.4		**Pixel 9 & 8 Alpha**
W4.3		**Pixel 7 & 6 Alpha**
W4.2		**Pixel 5 & 4 Alpha**
W4.1		**Pixel 3 & 2 Alpha**
W4.0		**Pixel 1 & 0 Alpha**

DWord	Bit	Description
W5.7		**Pixel 31 & 30 Alpha**
W5.6		**Pixel 29 & 28 Alpha**
W5.5		**Pixel 27 & 26 Alpha**
W5.4		**Pixel 25 & 24 Alpha**
W5.3		**Pixel 23 & 22 Alpha**
W5.2		**Pixel 21 & 20 Alpha**
W5.1		**Pixel 19 & 18 Alpha**
W5.0		**Pixel 17 & 16 Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:24	**Pixel 31 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 30 Red**
	15:8	**Pixel 29 Red**
	7:0	**Pixel 28 Red**
W0.6		**Pixel 27:24 Red**
W0.5		**Pixel 23:20 Red**
W0.4		**Pixel 19:16 Red**
W0.3		**Pixel 15:12 Red**
W0.2		**Pixel 11:8 Red**

DWord	Bit	Description
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1		**Pixels 31:0 Green**
W2		**Pixels 31:0 Blue**
W3		**Pixels 31:0 Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

If either red or blue channel (but not both) are masked, the W0 register is included in the payload but the masked channel is not written to the GRF. If both are masked, W0 is not included in the payload (reducing the response length by one).

DWord	Bit	Description
W0.7	31:24	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	23:16	**Pixel 28 Red**
	15:8	**Pixel 26 Red**
	7:0	**Pixel 24 Red**
W0.6		**Pixel 22, 20, 18, 16 Red**
W0.5		**Pixel 14, 12, 10, 8 Red**
W0.4		**Pixel 6, 4, 2, 0 Red**
W0.3		**Pixel 30, 28, 26, 24 Blue**
W0.2		**Pixel 22, 20, 18, 16 Blue**
W0.1		**Pixel 14, 12, 10, 8 Blue**

DWord	Bit	Description
W0.0		**Pixel 6, 4, 2, 0 Blue**
W1.7	31:24	**Pixel 31 Green**
	23:16	**Pixel 30 Green**
	15:8	**Pixel 29 Green**
	7:0	**Pixel 28 Green**
W1.6		**Pixel 27:24 Green**
W1.5		**Pixel 23:20 Green**
W1.4		**Pixel 19:16 Green**
W1.3		**Pixel 15:12 Green**
W1.2		**Pixel 11:8 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2.7		**Pixel 31:28 Alpha**
W2.6		**Pixel 27:24 Alpha**
W2.5		**Pixel 23:20 Alpha**
W2.4		**Pixel 19:16 Alpha**
W2.3		**Pixel 15:12 Alpha**
W2.2		**Pixel 11:8 Alpha**
W2.1		**Pixel 7:4 Alpha**
W2.0		**Pixel 3:0 Alpha**

Additional Writeback Phase for sample_unorm+killpix

For the sample_unorm+killpix messages, an additional writeback phase is returned. The value of "n" depends on which channels are enabled for return and the **Output Format Control Mode**, this register will immediately follow the first part of the writeback message.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>Wn.7:1</td><td></td><td>Reserved (not written)</td></tr>
<tr><td>Wn.0</td><td>31:0</td><td>Active Pixel Mask: This field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode.

The bits in this mask correspond to the pixels as follows and they are listed from upper left (MSB) lower right LSB:

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td></tr>
<tr><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

Cache_flush

The writeback message is for cache_flush indicates that the flush has been completed. The destination register is not modified.

DWord	Bit	Description
W0.7:0		Reserved

Sample_8x8 Writeback Messages

The writeback message for sample_8x8 consists of up to 16 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in all four destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel.

Pixels are numbered as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

"16 bit Full" Output Format Control Mode

DWord	Bits	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 11 & 10 Red**
W0.4		**Pixel 9 & 8 Red**
W0.3		**Pixel 7 & 6 Red**
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1		**Pixel 31:16 Red**
W2		**Pixels 47:32 Red**
W3		**Pixels 63:33 Red**
W4		**Pixels 15:0 Green**
W5		**Pixels 31:16 Green**
W6		**Pixels 47:32 Green**
W7		**Pixels 63:33 Green**
W8		**Pixels 15:0 Blue**
W9		**Pixels 31:16 Blue**
W10		**Pixels 47:32 Blue**

DWord	Bits	Description
W11		**Pixels 63:33 Blue**
W12		**Pixels 15:0 Alpha**
W13		**Pixels 31:16 Alpha**
W14		**Pixels 47:32 Alpha**
W15		**Pixels 63:33 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bits	Description
W0.7	31:16	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 28 Red**
W0.6		**Pixel 26 & 24 Red**
W0.5		**Pixel 22 & 20 Red**
W0.4		**Pixel 18 & 16 Red**
W0.3		**Pixel 14 & 12 Red**
W0.2		**Pixel 10 & 8 Red**
W0.1		**Pixel 6 & 4 Red**
W0.0		**Pixel 2 & 0 Red**
W1.7		**Pixel 62 & 60 Red**
W1.6		**Pixel 58 & 56 Red**
W1.5		**Pixel 54 & 52 Red**

DWord	Bits	Description
W1.4		**Pixel 50 & 48 Red**
W1.3		**Pixel 46 & 44 Red**
W1.2		**Pixel 42 & 40 Red**
W1.1		**Pixel 38 & 36 Red**
W1.0		**Pixel 34 & 32 Red**
W2.7	31:16	**Pixel 15 Green**
	15:0	**Pixel 14 Green**
W2.6		**Pixel 13 & 12 Green**
W2.5		**Pixel 11 & 10 Green**
W2.4		**Pixel 9 & 8 Green**
W2.3		**Pixel 7 & 6 Green**
W2.2		**Pixel 5 & 4 Green**
W2.1		**Pixel 3 & 2 Green**
W2.0		**Pixel 1 & 0 Green**
W3		**Pixel 31:16 Green**
W4		**Pixel 47:32 Green**
W5		**Pixel 63:48 Green**
W6.7	31:16	**Pixel 30 Blue**
	15:0	**Pixel 28 Blue**
W6.6		**Pixel 26 & 24 Blue**
W6.5		**Pixel 22 & 20 Blue**

DWord	Bits	Description
W6.4		**Pixel 18 & 16 Blue**
W6.3		**Pixel 14 & 12 Blue**
W6.2		**Pixel 10 & 8 Blue**
W6.1		**Pixel 6 & 4 Blue**
W6.0		**Pixel 2 & 0 Blue**
W7.7		**Pixel 62 & 60 Blue**
W7.6		**Pixel 58 & 56 Blue**
W7.5		**Pixel 54 & 52 Blue**
W7.4		**Pixel 50 & 48 Blue**
W7.3		**Pixel 46 & 44 Blue**
W7.2		**Pixel 42 & 40 Blue**
W7.1		**Pixel 38 & 36 Blue**
W7.0		**Pixel 34 & 32 Blue**
W8.7	31:16	**Pixel 15 Alpha**
	15:0	**Pixel 14 Alpha**
W8.6		**Pixel 13 & 12 Alpha**
W8.5		**Pixel 11 & 10 Alpha**
W8.4		**Pixel 9 & 8 Alpha**
W8.3		**Pixel 7 & 6 Alpha**
W8.2		**Pixel 5 & 4 Alpha**
W8.1		**Pixel 3 & 2 Alpha**

DWord	Bits	Description
W8.0		**Pixel 1 & 0 Alpha**
W9		**Pixel 31:16 Alpha**
W10		**Pixel 47:32 Alpha**
W11		**Pixel 63:48 Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 31 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 30 Red**
	15:8	**Pixel 29 Red**
	7:0	**Pixel 28 Red**
W0.6		**Pixel 27:24 Red**
W0.5		**Pixel 23:20 Red**
W0.4		**Pixel 19:16 Red**
W0.3		**Pixel 15:12 Red**
W0.2		**Pixel 11:8 Red**
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1.7		**Pixel 63:60 Red**
W1.6		**Pixel 59:56 Red**
W1.5		**Pixel 55:52 Red**

DWord	Bits	Description
W1.4		**Pixel 51:48 Red**
W1.3		**Pixel 47:44 Red**
W1.2		**Pixel 43:40 Red**
W1.1		**Pixel 39:36 Red**
W1.0		**Pixel 35:52 Red**
W2		**Pixels 31:0 Green**
W3		**Pixels 63:32 Green**
W4		**Pixels 31:0 Blue**
W5		**Pixels 63:32 Blue**
W6		**Pixels 31:0 Alpha**
W7		**Pixels 63:32 Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 62 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 60 Red**
	15:8	**Pixel 58 Red**
	7:0	**Pixel 56 Red**
W0.6		**Pixel 54, 52, 50, 48 Red**
W0.5		**Pixel 46, 44, 42, 40 Red**
W0.4		**Pixel 38, 36, 34, 32 Red**

DWord	Bits	Description
W0.3		**Pixel 30, 28, 26, 24 Red**
W0.2		**Pixel 22, 20, 18, 16 Red**
W0.1		**Pixel 14, 12, 10, 8 Red**
W0.0		**Pixel 6, 4, 2, 0 Red**
W1.7	31:24	**Pixel 31 Green**
	23:16	**Pixel 30 Green**
	15:8	**Pixel 29 Green**
	7:0	**Pixel 28 Green**
W1.6		**Pixel 27:24 Green**
W1.5		**Pixel 23:20 Green**
W1.4		**Pixel 19:16 Green**
W1.3		**Pixel 15:12 Green**
W1.2		**Pixel 11:8 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2		**Pixel 63:32 Green**
W3.7	31:24	**Pixel 62 Blue**
	23:16	**Pixel 60 Blue**
	15:8	**Pixel 58 Blue**
	7:0	**Pixel 56 Blue**
W3.6		**Pixel 54, 52, 50, 48 Blue**

DWord	Bits	Description
W3.5		**Pixel 46, 44, 42, 40 Blue**
W3.4		**Pixel 38, 36, 34, 32 Blue**
W3.3		**Pixel 30, 28, 26, 24 Blue**
W3.2		**Pixel 22, 20, 18, 16 Blue**
W3.1		**Pixel 14, 12, 10, 8 Blue**
W3.0		**Pixel 6, 4, 2, 0 Blue**
W4.7	31:24	**Pixel 31 Alpha**
	23:16	**Pixel 30 Alpha**
	15:8	**Pixel 29 Alpha**
	7:0	**Pixel 28 Alpha**
W4.6		**Pixel 27:24 Alpha**
W4.5		**Pixel 23:20 Alpha**
W4.4		**Pixel 19:16 Alpha**
W4.3		**Pixel 15:12 Alpha**
W4.2		**Pixel 11:8 Alpha**
W4.1		**Pixel 7:4 Alpha**
W4.0		**Pixel 3:0 Alpha**
W5		**Pixel 63:32 Alpha**

Sampler_8x8 – Writeback Message for AVS

Output format when "Shuffle_OutputWriteback=1" for 8x4 and 16x4 messages

For 16x4 message :

MSB															LSB
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48

For 8x4 message (though the message is only 32pixel, the below numbers are used to map to the same writeback format):

MSB							LSB
7	6	5	4	3	2	1	0
23	22	21	20	19	18	17	16
39	38	37	36	35	34	33	32
55	54	53	52	51	50	49	48

"16 bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:16	**Pixel 23 Red** **Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0)** **Range = [0000h:FF00h]**
	15:0	**Pixel 22 Red**
W0.6		**Pixel 21 & 20 Red**
W0.5		**Pixel 19 & 18 Red**
W0.4		**Pixel 17 & 16 Red**
W0.3		**Pixel 7 & 6 Red**
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 55 & 54 Red**
W1.6		**Pixel 53 & 52 Red**
W1.5		**Pixel 51 & 50 Red**
W1.4		**Pixel 49 & 48 Red**
W1.3		**Pixel 39 & 38 Red**
W1.2		**Pixel 37 & 36 Red**
W1.1		**Pixel 35 & 34 Red**
W1.0		**Pixel 33 & 32 Red**
W2		**Pixels 23:16, 7:0 Green**

DWord	Bit	Description
W3		**Pixels 55:48, 39:32 Green**
W4		**Pixels 23:16, 7:0 Blue**
W5		**Pixels 55:48, 39:32 Blue**
W6		**Pixels 23:16, 7:0 Alpha**
W7		**Pixels 55:48, 39:32 Alpha**
Below are valid only for 16x4 messages.		
W8		**Pixels 31:24, 15:8 Red**
W9		**Pixels 63:56, 47:40 Red**
W10		**Pixels 31:24, 15:8 Green**
W11		**Pixels 63:56, 47:40 Green**
W12		**Pixels 31:24, 15:8 Blue**
W13		**Pixels 63:56, 47:40 Blue**
W14		**Pixels 31:24, 15:8 Alpha**
W15		**Pixels 63:56, 47:40 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bit	Description
W0.7	31:16	**Pixel 54 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 52 Red**
W0.6		**Pixel 50 & 48 Red**
W0.5		**Pixel 38 & 36 Red**
W0.4		**Pixel 34 & 32 Red**
W0.3		**Pixel 22 & 20 Red**
W0.2		**Pixel 18 & 16 Red**
W0.1		**Pixel 6 & 4 Red**

DWord	Bit	Description
W0.0		**Pixel 2 & 0 Red**
W1.7	31:16	**Pixel 23 Green**
	15:0	**Pixel 22 Green**
W1.6		**Pixel 21 & 20 Green**
W1.5		**Pixel 19 & 18 Green**
W1.4		**Pixel 17 & 16 Green**
W1.3		**Pixel 7 & 6 Green**
W1.2		**Pixel 5 & 4 Green**
W1.1		**Pixel 3 & 2 Green**
W1.0		**Pixel 1 & 0 Green**
W2.7		**Pixel 55 & 54 Green**
W2.6		**Pixel 53 & 52 Green**
W2.5		**Pixel 51 & 50 Green**
W2.4		**Pixel 49 & 48 Green**
W2.3		**Pixel 39 & 38 Green**
W2.2		**Pixel 37 & 36 Green**
W2.1		**Pixel 35 & 34 Green**
W2.0		**Pixel 33 & 32 Green**
W3.7	31:16	**Pixel 54 Blue**
	15:0	**Pixel 52 Blue**
W3.6		**Pixel 50 & 48 Blue**

DWord	Bit	Description
W3.5		**Pixel 38 & 36 Blue**
W3.4		**Pixel 34 & 32 Blue**
W3.3		**Pixel 22 & 20 Blue**
W3.2		**Pixel 18 & 16 Blue**
W3.1		**Pixel 6 & 4 Blue**
W3.0		**Pixel 2 & 0 Blue**
W4.7	31:16	**Pixel 23 Alpha**
	15:0	**Pixel 22 Alpha**
W4.6		**Pixel 21 & 20 Alpha**
W4.5		**Pixel 19 & 18 Alpha**
W4.4		**Pixel 17 & 16 Alpha**
W4.3		**Pixel 7 & 6 Alpha**
W4.2		**Pixel 5 & 4 Alpha**
W4.1		**Pixel 3 & 2 Alpha**
W4.0		**Pixel 1 & 0 Alpha**
W5.7		**Pixel 55 & 54 Alpha**
W5.6		**Pixel 53 & 52 Alpha**
W5.5		**Pixel 51 & 50 Alpha**
W5.4		**Pixel 49 & 48 Alpha**
W5.3		**Pixel 39 & 38 Alpha**
W5.2		**Pixel 37 & 36 Alpha**

DWord	Bit	Description
W5.1		**Pixel 35 & 34 Alpha**
W5.0		**Pixel 33 & 32 Alpha**
Below DWs are only valid for 16x4 message		
W6.7	31:48	**Pixel 62 Red**
	15:0	**Pixel 60 Red**
W6.6		**Pixel 58 & 56 Red**
W6.5		**Pixel 46 & 44 Red**
W6.4		**Pixel 42 & 40 Red**
W6.3		**Pixel 30 & 28 Red**
W6.2		**Pixel 26 & 24 Red**
W6.1		**Pixel 14 & 12 Red**
W6.0		**Pixel 10 & 8 Red**
W7.7	31:48	**Pixel 63 Green**
	15:0	**Pixel 62 Green**
W7.6		**Pixel 61 & 60 Green**
W7.5		**Pixel 59 & 58 Green**
W7.4		**Pixel 57 & 56 Green**
W7.3		**Pixel 47 & 46 Green**
W7.2		**Pixel 45 & 44 Green**
W7.1		**Pixel 43 & 42 Green**
W7.0		**Pixel 41 & 40 Green**

DWord	Bit	Description
W8.7		**Pixel 31 & 30 Green**
W8.6		**Pixel 29 & 28 Green**
W8.5		**Pixel 27 & 26 Green**
W8.4		**Pixel 25 & 24 Green**
W8.3		**Pixel 15 & 14 Green**
W8.2		**Pixel 13 & 12 Green**
W8.1		**Pixel 11 & 10 Green**
W8.0		**Pixel 9 & 8 Green**
W9.7	31:48	**Pixel 62 Blue**
	15:0	**Pixel 60 Blue**
W9.6		**Pixel 58 & 56 Blue**
W9.5		**Pixel 46 & 44 Blue**
W9.4		**Pixel 42 & 40 Blue**
W9.3		**Pixel 30 & 28 Blue**
W9.2		**Pixel 26 & 24 Blue**
W9.1		**Pixel 14 & 12 Blue**
W9.0		**Pixel 10 & 8 Blue**
W10.7	31:48	**Pixel 63 Alpha**
	15:0	**Pixel 62 Alpha**
W10.6		**Pixel 61 & 60 Alpha**
W10.5		**Pixel 59 & 58 Alpha**

DWord	Bit	Description
W10.4		**Pixel 57 & 56 Alpha**
W10.3		**Pixel 47 & 46 Alpha**
W10.2		**Pixel 45 & 44 Alpha**
W10.1		**Pixel 43 & 42 Alpha**
W10.0		**Pixel 41 & 40 Alpha**
W11.7		**Pixel 31 & 30 Alpha**
W11.6		**Pixel 29 & 28 Alpha**
W11.5		**Pixel 27 & 26 Alpha**
W11.4		**Pixel 25 & 24 Alpha**
W11.3		**Pixel 15 & 14 Alpha**
W11.2		**Pixel 13 & 12 Alpha**
W11.1		**Pixel 11 & 10 Alpha**
W11.0		**Pixel 9 & 8 Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:24	**Pixel 55 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 54 Red**
	15:8	**Pixel 53 Red**
	7:0	**Pixel 52 Red**
W0.6		**Pixel 51:48 Red**

DWord	Bit	Description
W0.5		**Pixel 39:36 Red**
W0.4		**Pixel 35:32 Red**
W0.3		**Pixel 23:20 Red**
W0.2		**Pixel 19:16 Red**
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1		**Pixels [55:48],[39:32],[23:16],[7:0] Green**
W2		**Pixels [55:48],[39:32],[23:16],[7:0] Blue**
W3		**Pixels [55:48],[39:32],[23:16],[7:0] Alpha**
Below DWs are only valid for 16x4 message		
W4.7	31:24	**Pixel 63 Red**
	23:16	**Pixel 62 Red**
	15:8	**Pixel 61 Red**
	7:0	**Pixel 60 Red**
W4.6		**Pixel 59:56 Red**
W4.5		**Pixel 47:44 Red**
W4.4		**Pixel 43:40 Red**
W4.3		**Pixel 31:28 Red**
W4.2		**Pixel 27:24 Red**
W4.1		**Pixel 15:12 Red**
W4.0		**Pixel 11:8 Red**

DWord	Bit	Description
W5		**Pixels [63:56],[47:40],[31:24],[15:8] Green**
W6		**Pixels [63:56],[47:40],[31:24],[15:8] Blue**
W7		**Pixels [63:56],[47:40],[31:24],[15:8] Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

If either red or blue channel (but not both) are masked, the W0 and W3 registers are included in the payload but the masked channel is not written to the GRF. If both are masked, W0 and W3 are not included in the payload (reducing the response length by two).

DWord	Bit	Description
W0.7	31:24	**Pixel 54 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	23:16	**Pixel 52 Red**
	15:8	**Pixel 50 Red**
	7:0	**Pixel 48 Red**
W0.6		**Pixel 38, 36, 34, 32 Red**
W0.5		**Pixel 22, 20, 18, 16 Red**
W0.4		**Pixel 6, 4, 2, 0 Red**
W0.3		**Pixel 54, 52, 50, 48 Blue**
W0.2		**Pixel 38, 36, 34, 32 Blue**
W0.1		**Pixel 22, 20, 18, 16 Blue**
W0.0		**Pixel 6, 4, 2, 0 Blue**
W1.7	31:24	**Pixel 55 Green**
	23:16	**Pixel 54 Green**

DWord	Bit	Description
	15:8	**Pixel 53 Green**
	7:0	**Pixel 52 Green**
W1.6		**Pixel 51:48 Green**
W1.5		**Pixel 39:36 Green**
W1.4		**Pixel 35:32 Green**
W1.3		**Pixel 23:20 Green**
W1.2		**Pixel 19:16 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2.7		**Pixel 55:52 Alpha**
W2.6		**Pixel 51:48 Alpha**
W2.5		**Pixel 39:36 Alpha**
W2.4		**Pixel 35:32 Alpha**
W2.3		**Pixel 23:20 Alpha**
W2.2		**Pixel 19:16 Alpha**
W2.1		**Pixel 7:4 Alpha**
W2.0		**Pixel 3:0 Alpha**
Below DWs are only valid for 16x4 message		
W3.7	31:24	**Pixel 62 Red**
	23:16	**Pixel 60 Red**
	15:8	**Pixel 58 Red**

DWord	Bit	Description
	7:0	**Pixel 56 Red**
W3.6		**Pixel 46, 44, 42, 40 Red**
W3.5		**Pixel 30, 28, 26, 24 Red**
W3.4		**Pixel 14, 12, 10, 8 Red**
W3.3		**Pixel 62, 60, 58, 56 Blue**
W3.2		**Pixel 46, 44, 42, 40 Blue**
W3.1		**Pixel 30, 28, 26, 24 Blue**
W3.0		**Pixel 14, 12, 10, 8 Blue**
W4.7	31:24	**Pixel 63 Green**
	23:16	**Pixel 62 Green**
	15:8	**Pixel 61 Green**
	7:0	**Pixel 60 Green**
W4.6		**Pixel 59:56 Green**
W4.5		**Pixel 47:44 Green**
W4.4		**Pixel 43:40 Green**
W4.3		**Pixel 31:28 Green**
W4.2		**Pixel 27:24 Green**
W4.1		**Pixel 15:12 Green**
W4.0		**Pixel 11:8 Green**
W5.7		**Pixel 63:60 Alpha**
W5.6		**Pixel 59:56 Alpha**

DWord	Bit	Description
W5.5		**Pixel 47:44 Alpha**
W5.4		**Pixel 43:40 Alpha**
W5.3		**Pixel 31:28 Alpha**
W5.2		**Pixel 27:24 Alpha**
W5.1		**Pixel 15:12 Alpha**
W5.0		**Pixel 11:8 Alpha**

SIMD32 Surface State

Please refer to the 3D Surface State definition in the SURFACE_STATE topic.

SIMD32 Sampler State

Please refer to the 3D Sampler State definition at *Sampler State.*

3D Pipeline Stages

The following table lists the various stages of the 3D pipeline and describes their major functions.

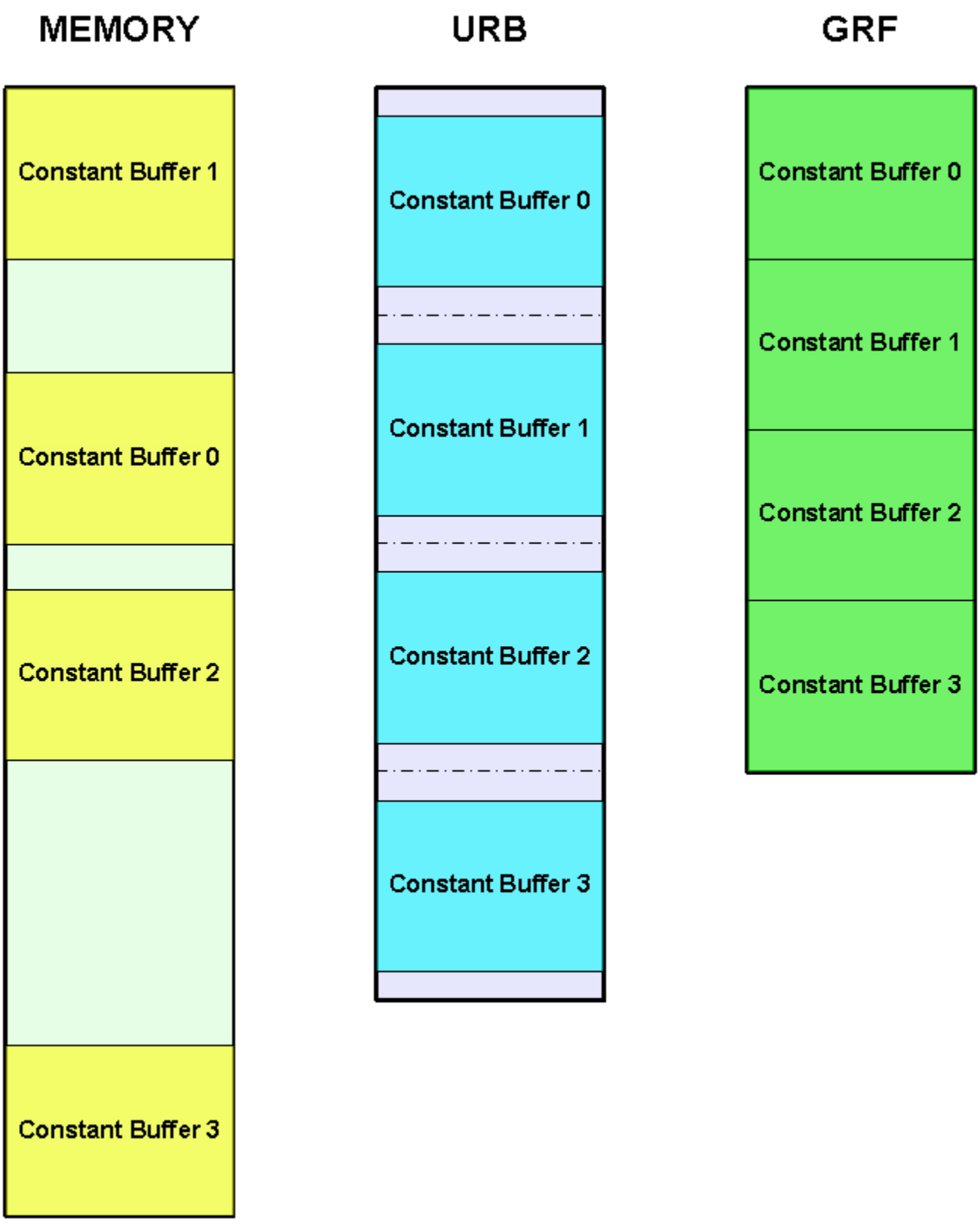
Pipeline Stage	Functions Performed
Command Stream (CS)	The Command Stream stage is responsible for managing the 3D pipeline and passing commands down the pipeline. In addition, the CS unit reads "constant data" from memory buffers and places it in the URB. Note that the CS stage is shared between the 3D, GPGPU and Media pipelines.
Vertex Fetch (VF)	The Vertex Fetch stage, in response to 3D Primitive Processing commands, is responsible for reading vertex data from memory, reformatting it, and writing the results into Vertex URB Entries. It then outputs primitives by passing references to the VUEs down the pipeline.
Vertex Shader (VS)	The Vertex Shader stage is responsible for processing (shading) incoming vertices by passing them to VS threads.
Hull Shader (HS)	The Hull Shader is responsible for processing (shading) incoming patch primitives as part of the tessellation process.
Tessellation Engine (TE)	The Tessellation Engine is responsible for using tessellation factors (computed in the HS stage) to tessellate U,V parametric domains into domain point topologies.
Domain Shader (DS)	The Domain Shader stage is responsible for processing (shading) the domain points (generated by the TE stage) into corresponding vertices.
Geometry Shader (GS)	The Geometry Shader stage is responsible for processing incoming objects by passing each object's vertices to a GS thread.
Stream Output Logic (SOL)	The Stream Output Logic is responsible for outputting incoming object vertices into Stream Out Buffers in memory.
Clipper (CLIP)	The Clipper stage performs Clip Tests on incoming objects and clips objects if required. Objects are clipped using fixed-function hardware.
Strip/Fan (SF)	The Strip/Fan stage performs object setup. Object setup uses fixed-function hardware.
Windower/Masker (WM)	The Windower/Masker performs object rasterization and determines visibility coverage.

3D Pipeline-Level State

This section contains table commands for the 3D Pipeline Level.

Programming Note	
Context:	3D Pipeline-Level State - Push Constant URB Allocation
The push constants are buffered in the Push Constant section of the URB which is part of the L3$. Software is required to program the hardware to allocate space in the URB for each shader push constant. The software is limited to the low addresses of the URB and must ensure that none of the shaders have overlapping handles. Software may use some if not all of the Push Constant region of the URB for pr-stage handle allocations as long as none of the push constants and handle allocations overlap. Refer to the various 3DSTATE_PUSH_CONSTANT_ALLOC_xx state commands for details regarding the maximum size of the Push Constant and other state programming information.	

Below is a diagram that represents how the hardware may move and store one CONSTANT_BUFFER command for a fixed function shader:



The bubbles in the URB are caused by the constant buffer in memory starting on a half cacheline and

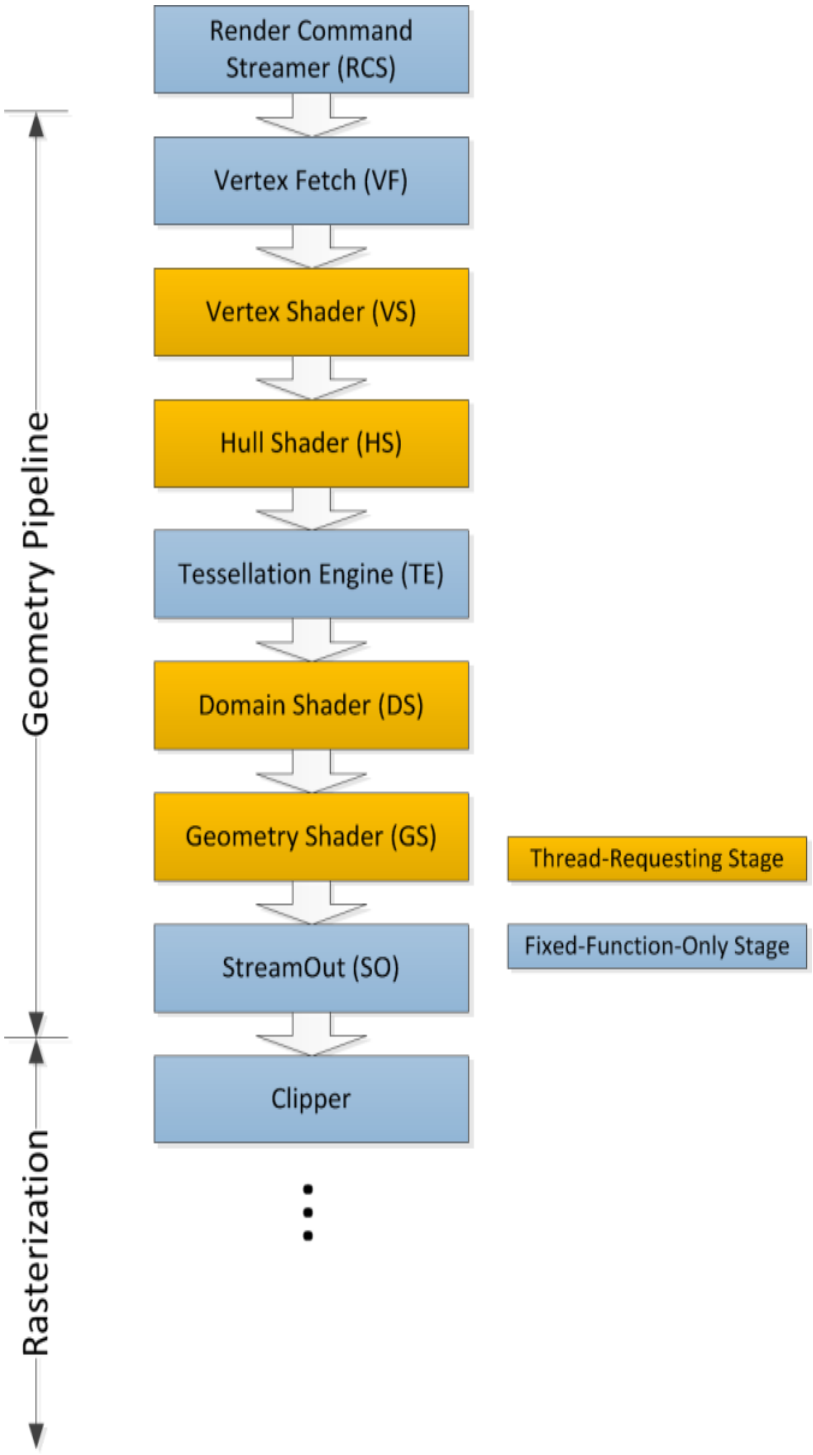
being an even number in length. If the constant buffer starts on an odd cacheline and has an odd number length then there will only be a bubble at the beginning of the buffer in the URB. If the constant buffer in memory starts on a cache line boundary and has an odd number length then the bubble will only be at the end of the constant buffer in the URB. Once the constant buffer is written to the GRF space then all the bubbles will be removed.

Software must guarantee that there is enough space in the push constant buffer in the URB to hold one constant buffer from memory. This includes any buffering to write the 512b aligned requests from memory into the URB. Because the L3$ only supports writes from memory in 512b chunks, the URB may have some bubbles between each constant buffer fetch.

3D Pipeline Geometry

Block Diagram

The following block diagram shows the stages of the Geometry Pipeline and where they are positioned in the overall 3D Pipeline.


Render Command Streamer (RCS)
Vertex Fetch (VF)
Vertex Shader (VS)
Hull Shader (HS)
Tessellation Engine (TE)
Domain Shader (DS)
Geometry Shader (GS)
StreamOut (SO)
Clipper
Geometry Pipeline
Rasterization
Thread-Requesting Stage
Fixed-Function-Only Stage

3D Primitives Overview

The 3DPRIMITIVE command (defined in the VF Stage chapter) is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in the rendering of pixel data into the render targets, but this is not required.

There is considerable confusion surrounding the term 'primitive', e.g., is a triangle strip a 'primitive', or is a triangle within a triangle strip a 'primitive'? Some APIs use the term 'topology' to describe the higher-level construct (e.g., a triangle strip), and uses the term 'primitive' when discussing a triangle within a triangle strip. In this spec, we will try to avoid ambiguity by using the term 'object' to represent the basic shapes (point, line, triangle), and 'topology' to represent input geometry (strips, lists, etc.). Unfortunately, terms like '3DPRIMITIVE' must remain for legacy reasons.

The following table describes the basic primitive topology types supported in the 3D pipeline.

Programming Note	
Context:	3D Primitives Overview
• There are several variants of the basic topologies. These have been introduced to allow slight variations in behavior without requiring a state change. • Number of vertices and Dangling Vertices: Topologies have an "expected" number of vertices in order to form complete objects within the topologies (e.g., LINELIST is expected to have an even number of vertices). The actual number of vertices specified in the 3DPRIMITIVE command, and as output from the GS unit, is allowed to deviate from this expected number, in which case any "dangling" vertices are discarded. The removal of dangling vertices is initially performed in the VF unit. To filter out dangling vertices emitted by GS threads, the CLIP unit also performs dangling-vertex removal at its input.	

3D Primitive Topology Types

3D Primitive Topology Type (ordered alphabetically)	Description
LINELIST	• A list of independent line objects (2 vertices per line). • Normal usage expects a multiple of 2 vertices, though incomplete objects are silently ignored.
LINELIST_ADJ	• A list of independent line objects with adjacency information (4 vertices per line). • Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
LINELOOP	• Similar to a 3DPRIM_LINESTRIP, though the last vertex is connected back to the

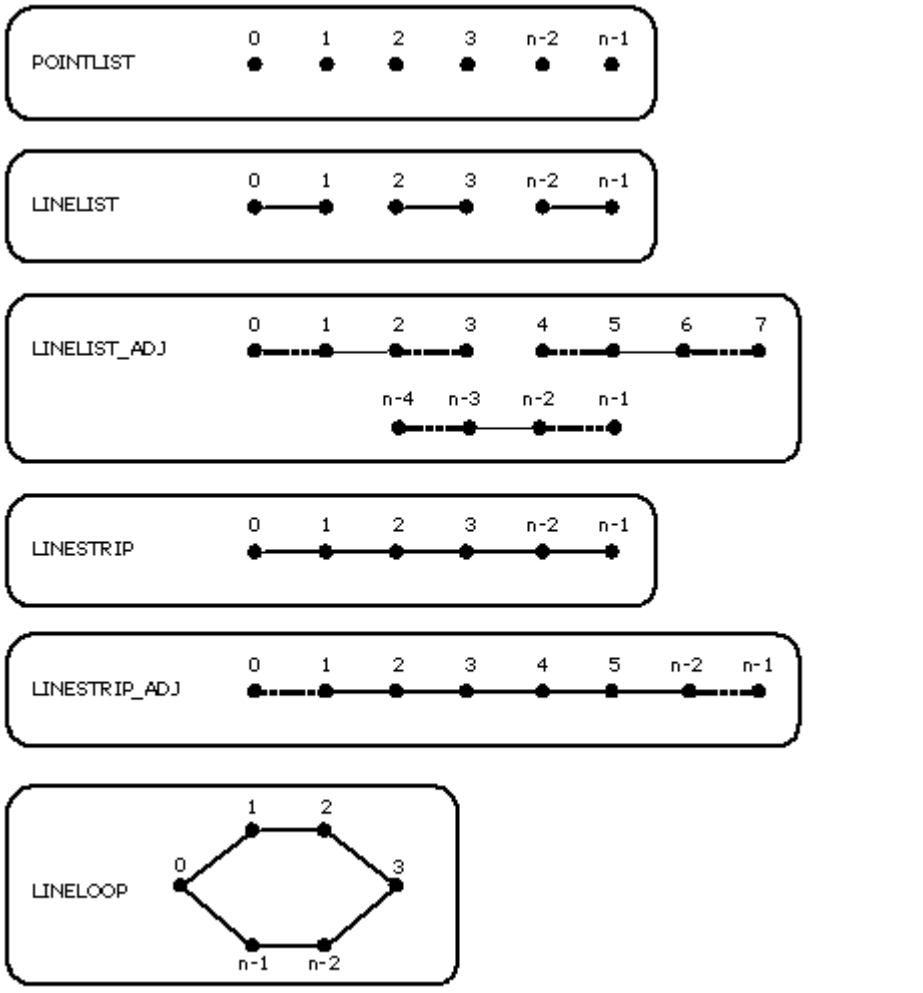
3D Primitive Topology Type (ordered alphabetically)	Description
	initial vertex via a line object. The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline. • Normal usage expects at least 2 vertices, though incomplete objects are silently ignored. (The 2-vertex case is required by OGL). • Not valid after the GS stage (i.e., must be converted by a GS thread to some other primitive type).
LINESTRIP	• A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define a connected line object. • Normal usage expects at least 2 vertices, though incomplete objects are silently ignored.
LINESTRIP_ADJ	• A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define connected line object. The first and last segments are adjacent–only vertices. • Normal usage expects at least 4 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
LINESTRIP_BF	• Similar to LINESTRIP, except treated as “backfacing’ during rasterization (stencil test). • This can be used to support “line” polygon fill mode when two-sided stencil is enabled.
LINESTRIP_CONT	• Similar to LINESTRIP, except LineStipple (if enabled) is continued (vs. reset) at the start of the primitive topology. • This can be used to support line stipple when the API-provided primitive is split across multiple tolopologies.
LINESTRIP_CONT_BF	Combination of LINESTRIP_BF and LINESTRIP_CONT variations.
POINTLIST	A list of point objects (1 vertex per point).
POINTLIST_BF	• Similar to POINTLIST, except treated as “backfacing’ during rasterization (stencil test). • This can be used to support “point” polygon fill mode when two-sided stencil is enabled.
POLYGON	• Similar to TRIFAN, though the first vertex always provides the “flat-shaded” values (vs. this being programmable through state). • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.

3D Primitive Topology Type (ordered alphabetically)	Description
QUADLIST	• A list of independent quad objects (4 vertices per quad). • The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline. • Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored.
QUADSTRIP	• A list of vertices connected such that, after the first two vertices, each additional pair of vertices are associated with the previous two vertices to define a connected quad object. • Normal usage expects an even number (4 or greater) of vertices, though incomplete objects are silently ignored.
RECTLIST	• A list of independent rectangles, where only 3 vertices are provided per rectangle object, with the fourth vertex implied by the definition of a rectangle. V0=LowerRight, V1=LowerLeft, V2=UpperLeft. Implied V3 = V0-V1+V2. • Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored. • The RECTLIST primitive is supported specifically for 2D operations (e.g., BLTs and "stretch" BLTs) and not as a general 3D primitive. Due to this, a number of restrictions apply to the use of RECTLIST: • Must utilize "screen space" coordinates (VPOS_SCREENSPACE) when the primitive reaches the CLIP stage. The W component of position must be 1.0 for all vertices. The 3 vertices of each object should specify a screen-aligned rectangle (after the implied vertex is computed). • Clipping: Must not require clipping or rely on the CLIP unit's ClipTest logic to determine if clipping is required. Either the CLIP unit should be DISABLED, or the CLIP unit's Clip Mode should be set to a value other than CLIPMODE_NORMAL. • Viewport Mapping must be DISABLED (as is typical with the use of screen-space coordinates).
RECTLIST_SUBPIXEL	The subpixel precise, axis-aligned bounding box of the object's 3 vertices is rendered.
TRIFAN	• Triangle objects arranged in a fan (or polygon). The initial vertex is maintained as a common vertex. After the second vertex, each additional vertex is associated with the previous vertex and the common vertex to define a connected triangle object. • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
TRIFAN_NOSTIPPLE	• Similar to TRIFAN, but poylgon stipple is not applied (even if enabled). • This can be used to support "point" polygon fill mode, under the combination

3D Primitive Topology Type (ordered alphabetically)	Description
	of the following conditions: (a) when the frontfacing and backfacing polygon fill modes are different (so the final fill mode is not known to the driver), (b) one of the fill modes is “point” and the other is “solid”, (c) point mode is being emulated by converting the point into a trifan, (d) polygon stipple is enabled. In this case, polygon stipple should not be applied to the points-emulated-as-trifans.
TRILIST	• A list of independent triangle objects (3 vertices per triangle). • Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored.
TRILIST_ADJ	• A list of independent triangle objects with adjacency information (6 vertices per triangle). • Normal usage expects a multiple of 6 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
TRISTRIP	• A list of vertices connected such that, after the first two vertices, each additional vertex is associated with the last two vertices to define a connected triangle object. • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
TRISTRIP_ADJ	• A list of vertices where the even-numbered (including 0th) vertices are connected such that, after the first two vertex pairs, each additional even-numbered vertex is associated with the last two even-numbered vertices to define a connected triangle object. The odd-numbered vertices are adjacent-only vertices. VFUNIT will complete a drawcall with the topology of tristrip_adj even if there is a preemption request in the middle of the draw call. • Normal usage expects at least 6 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
TRISTRIP_REVERSE	Similar to TRISTRIP, though the sense of orientation (winding order) is reversed – this allows SW to break long tristrips into smaller pieces and still maintain correct face orientations.

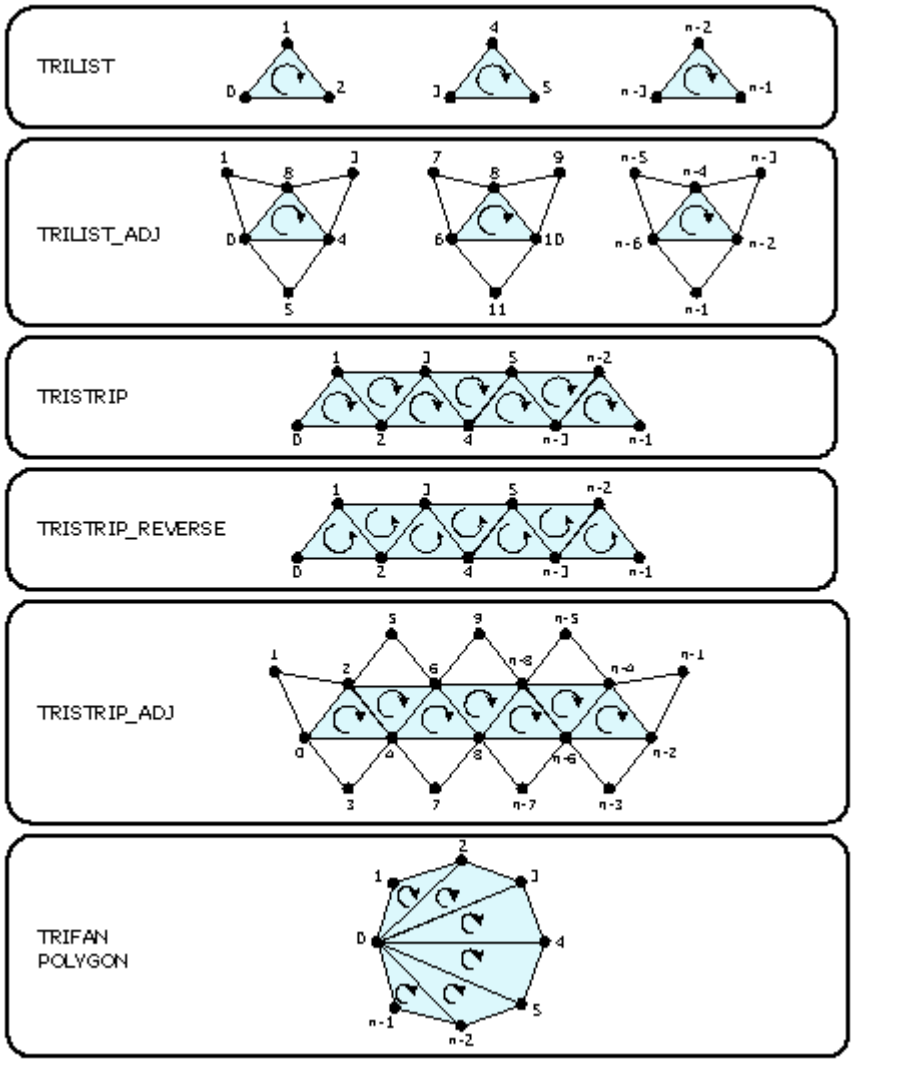
3D Primitive Topology Type (ordered alphabetically)	Description
: PATCHLIST_n	List of n-vertex “patch” objects. These topologies cannot be rendered directly – the tessellation units must be used to convert them into points, lines, or triangles to produce rasterization results. (VS, GS, and StreamOutput operations can also be performed.)

The following diagrams illustrate the basic 3D primitive topologies. (Variants are not shown if they have the same definition with respect to the information provided in the diagrams).

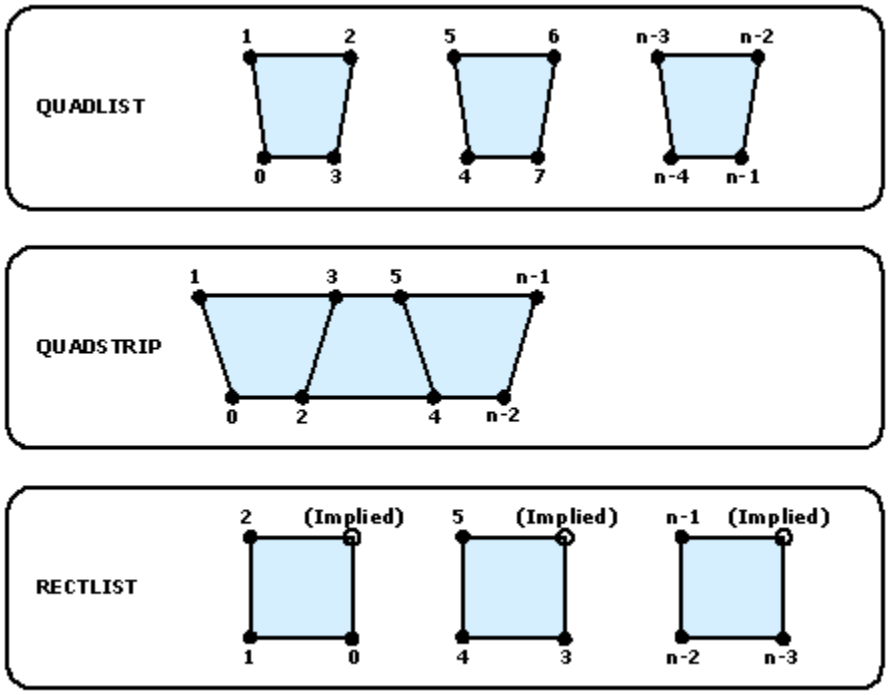


B6815-01

A note on the arrows you see below: These arrows are intended to show the vertex ordering of triangles that are to be considered having “clockwise” winding order in screen space. Effectively, the arrows show the order in which vertices are used in the cross-product (area, determinant) computation. Note that for TRISTRIP, this requires that either the order of odd-numbered triangles be reversed in the cross-product or the sign of the result of the normally-ordered cross-product be flipped (these are identical operations).


TRILIST
TRILIST_ADJ
TRISTRIP
TRISTRIP_REVERSE
TRISTRIP_ADJ
TRIFAN
POLYGON

B6816-01


QUADLIST
1
2
0
3
5
6
4
7
n-3
n-2
n-4
n-1
QUADSTRIP
1
3
5
n-1
0
2
4
n-2
RECTLIST
2
(Implied)
1
0
5
(Implied)
4
3
n-1
(Implied)
n-2
n-3

B6818-01

Thread Request Generation

Once a FF unit determines that a thread can be requested, it must gather all the information required to submit the thread request to the Thread Dispatcher. This information is divided into several categories, as listed below and subsequently described in detail.

- **Thread Control Information:** This is the information required (from the FF unit) to establish the execution environment of the thread.
- **Thread Payload Header:** This is the first portion of the thread payload passed in the GRF, starting at GRF R0. This is information passed directly from the FF unit. It precedes the Thread Payload Input URB Data.
- **Thread Payload Input URB Data:** This is the second portion of the thread payload. It is read from the URB using entry handles supplied by the FF unit.

Thread Control Information

The following table describes the various state variables that a FF unit uses to provide information to the Thread Dispatcher and which affect the thread execution environment. Note that this information is not directly passed to the thread in the thread payload (though some fields may be subsequently accessed by the thread via architectural registers).

State Variables Included in Thread Control Information

State Variable	Usage	FFs
Kernel Start Pointer	This field, together with the **General State Pointer**, specifies the starting location (1st GEN4 core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the **General State Pointer**.	All FFs spawning threads
GRF Register Block Count	Specifies, in 16-register blocks, how many GRF registers are required to run the kernel. The Thread Dispatcher will only seek candidate EUs that have a sufficient number of GRF register blocks available. Upon selecting a target EU, the Thread DIspatcher will generate a logical-to-physical GRF mapping and provide this to the target EU.	All FFs spawning threads
Single Program Flow (SPF)	Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). See CR0 description in *ISA Execution Environment*.	All FFs spawning threads
Thread Dispatch Priority	The Thread Dispatcher will give priority to those thread requests with Thread Dispatch Priority of HIGH_PRIORITY over those marked as LOW_PRIORITY. Within these two classes of thread requests, the Thread Dispatcher applies a priority order (e.g., round-robin --- though this algorithm is considered a device implementation-dependent detail).	All FFs spawning threads
Floating	This determines the initial value of the **Floating Point Mode** bit of the EU's CR0	All FFs spawning

State Variable	Usage	FFs
Point Mode	architectural register that controls floating point behavior in the EU core. (See ISA.)	threads
Exceptions Enable	This bitmask controls the exception handing logic in the EU. (See ISA.)	All FFs spawning threads
Sampler Count	This is a hint which specifies how many indirect SAMPLER_STATE structures should be prefetched concurrent with thread initiation. It is recommended that software program this field to equal the number of samplers, though there may be some minor performance impact if this number gets large. This value should not exceed the number of samplers accessed by the thread as there would be no performance advantage. Note that the data prefetch is treated as any other memory fetch (with respect to page faults, etc.).	All stages supporting sampling (VS, GS, WM)
Binding Table Entry Count	This is a hint which specifies how many indirect BINDING_TABLE_STATE structures should be prefetched concurrent with thread initiation. (The notes included in Sampler Count (above) also apply to this field).	All FFs spawning threads

Thread Payload Generation

FF units are responsible for generating a thread *payload* – the data pre-loaded into the target EU's GRF registers (starting at R0) that serves as the primary direct input to a thread's kernel. The general format of these payloads follow a similar structure, though the exact payload size/content/layout is unique to each stage. This subsection describes the common aspects – refer to the specific stage's chapters for details on any differences.

The payload data is divided into two main sections: the *payload header* followed by the *payload URB data*. The payload header contains information passed directly from the FF unit, while the payload URB data is obtained from URB locations specified by the FF unit.

The first 256 bits of the thread payload (the initial contents of R0, aka "the R0 header") is specially formatted to closely match (and in some cases exactly match) the first 256 bits of thread-generated *messages* (i.e., the message header) accepted by shared functions. In fact, the send instruction supports having a copy of a GR's contents (such as R0) used as the message header. Software must take this intention into account (i.e., "don't muck with R0 unless you know what you're doing"). This is especially important given the fact that several fields in the R0 header are considered opaque to SW, where use or modification of their contents might lead to UNDEFINED results.

The payload header is further (loosely) divided into a leading *fixed payload header* section and a trailing, variable-sized *extended payload header* section. In general the size, content and layout of both payload header sections are FF-specific, though many of the fixed payload header fields are common amongst the FF stages. The extended header is used by the FF unit to pass additional information specific to that

FF unit. The extended header is defined to start after the fixed payload header and end at the offset defined by **Dispatch GRF Start Register for URB Data**. Software can cause use the **Dispatch GRF Start Register for URB Data** field to insert padding into the extended header in order to maintain a fixed offset for the start of the URB data.

Fixed Payload Header

The payload header is used to pass *FF pipeline information* required as thread input data. This information is a mixture of SW-provided state information (state table pointers, etc.), primitive information received by the FF unit from the FF pipeline, and parameters generated/computed by the FF unit. Most of the fields of the fixed header are common between the FF stages. These non-FF-specific fields are described in Fixed Payload Header Fields (non-FF-specific). Note that a particular stage's header may not contain all these fields, so they are not "common" in the strictest sense.

Fixed Payload Header Fields (non-FF-specific)

Fixed Payload Header Field (non-FF-specific)	Description	FFs
FF Unit ID	Function ID of the FF unit. This value identifies the FF unit within the GEN4 subsystem. The FF unit uses this field (when transmitted in a Message Header to the URB Function) to detect messages emanating from its spawned threads.	All FFs spawning threads
Snapshot Flag		All FFs spawning threads
Thread ID	This field uniquely identifies this thread within the FF unit over some period.	All FFs spawning threads
Scratch Space Pointer	This is the starting location of the thread's allocated scratch space, specified as an offset from the **General State Base Address**. Note that scratch space is allocated by the FF unit on a per-thread basis, based on the **Scratch Space Base Pointer** and **Per-Thread Scratch Space Size** state variables. FF units assign a thread an arbitrarily-positioned region within this space. The scratch space for multiple (API-visible) entities (vertices, pixels) is interleaved within the thread's scratch space.	All FFs spawning threads
Dispatch ID	This field identifies this thread within the outstanding threads spawned by the FF unit. This field does *not* uniquely identify the thread over any significant period. **Implementation Note:** This field is effectively an "active thread index". It is used on a thread's URB allocation request to identify which thread's handle pool is to source the allocation. It is used upon thread termination to free up the thread's scratch space allocation.	All FFs spawning threads
Binding Table Pointer	This field, together with the **Surface State Base Pointer**, specifies the starting location of the Binding Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **Surface State Base Pointer**. See *Shared Functions* for a description of a Binding Table.	All FFs spawning threads

Fixed Payload Header Field (non-FF-specific)	Description	FFs
Sampler State Pointer	This field, together with the **General State Base Pointer**, specifies the starting location of the Sampler State Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **General State Base Pointer**. See *Shared Functions* for a description of a Sampler State Table.	**All FFs spawning threads which sample (VS, GS, WM)**
Per Thread Scratch Space	This field specifies the amount of scratch space allocated to each thread spawned by the FF unit. The driver must allocate enough contiguous scratch space, starting at the **Scratch Space Base Pointer**, to ensure that the **Maximum Number of Threads** can each get **Per-Thread Scratch Space** size without exceeding the driver-allocated scratch space.	All FFs spawning threads
Handle ID <*n*>	This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. The FF unit will use this information upon detecting a URB_WRITE message issued by the thread. Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS, GS, CLIP, SF
URB Return Handle <*n*>	This is an initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the destination URB entry. Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS, GS, CLIP, SF
Primitive Topology Type	As part of processing an incoming primitive, a FF unit is often required to spawn a number of threads (for example, for each individual triangle in a TRIANGLE_STRIP). This field identifies the type of primitive which is being processed by the FF unit, and which has lead to the spawning of the thread. GEN4 kernels written to process different types of objects can use this value to direct that processing. E.g., when a CLIP kernel is to provide clipping for all the various primitive types, the kernel would need to examine the Primitive Topology Type to distinguish between point, lines, and triangle clipping requests. **Note:** In general, this field is identical to the Primitive Topology Type associated with the primitive vertices as received by the FF unit. Refer to the individual FF unit chapters for cases where the FF unit modifies the value before passing it to the thread. (for example, certain units perform toggling of TRIANGLESTRIP and TRIANGLESTRIP_REV).	GS, CLIP, SF, WM

Extended Payload Header

The extended header is of variable-size, where inclusion of a field is determined by FF unit state programming.

In order to permit the use of common kernels (thus reducing the number of kernels required), the **Dispatch GRF Start Register for URB Data** state variable is supported in all FF stages. This SV is used to place the payload URB data at a specific starting GRF register, irrespective of the size of the extended header. A kernel can therefore reference the payload URB data at fixed GRF locations, while conditionally referencing extended payload header information.

Payload URB Data

In each thread payload, following the payload header, is some amount of URB-sourced data required as input to the thread. This data is divided into an optional *Constant URB Entry* (CURBE), following either by a Primitive URB Entry (WM) or a number of Vertex URB Entries (VS, GS, CLIP, SF). A FF unit only knows the location of this data in the URB, and is never exposed to the contents. For each URB entry, the FF unit will supply a sequence of handles, read offsets and read lengths to the GEN4 subsystem. The subsystem will read the appropriate 256-bit locations of the URB, optionally perform swizzling (VS only), and write the results into sequential GRF registers (starting at **Dispatch GRF Start Register for URB Data**).

State Variables Controlling Payload URB Data

State Variable	Usage	FFs
Dispatch GRF Start Register for URB Data	This SV identifies the starting GRF register receiving payload URB data. Software is responsible for ensuring that URB data does not overwrite the Fixed or Extended Header portions of the payload.	FFs spawning threads
Vertex URB Entry Read Offset	This SV specifies the starting offset within VUEs from which vertex data is to be read and supplied in this stage's payloads. It is specified as a 256-bit offset into any and all VUEs passed in the payload. This SV can be used to skip over leading data in VUEs that is not required by the stage's threads (e.g., skipping over the Vertex Header data at the SF stage, as that information is not required for setup calculations). Skipping over irrelevant data can only help to improve performance. Specifying a vertex data source extending beyond the end of a vertex entry is UNDEFINED.	VS, GS
Vertex URB Entry Read Length	This SV determines the amount of vertex data (starting at **Vertex URB Entry Read Offset**) to be read from each VUEs and passed into the payload URB data. It is specified in 256-bit units. A zero value is INVALID (at very least one 256-bit unit must be read). Specifying a vertex data source extending beyond the end of a VUE is UNDEFINED.	

Programming Restrictions: (others may already been mentioned)

- The maximum size payload for any thread is limited by the number of GRF registers available to the thread, as determined by min(128, 16 * **GRF Register Block Count**). Software is responsible for ensuring this maximum size is not exceeded, taking into account:
  - The size of the Fixed and Extended Payload Header associated with the FF unit.
  - The **Dispatch GRF Start Register for URB** Data SV.
  - The amount of CURBE data included (via **Constant URB Entry Read Length**)
  - The number of VUEs included (as a function of FF unit, it's state programming, and incoming primitive types)
  - The amount of VUE data included for each vertex (via **Vertex URB Entry Read Length**)
  - (For WM-spawned PS threads) The amount of Primitive URB Entry data.
- For any type of URB Entry reads:
  - Specifying a source region (via Read Offset, Read Length) that goes past the end of the URB Entry allocation is illegal.
    - The allocated size of Vertex/Primitive URB Entries is determined by the **URB Entry Allocation Size** value provided in the pipeline state descriptor of the FF unit owning the VUE/PUE.
    - The allocated size of CURBE entries is determined by the **URB Entry Allocation Size** value provided in the CS_URB_STATE command.

Vertex Data Overview

The 3D pipeline FF stages (past VF) receive input 3D primitives as a stream of vertex information packets. (These packets are not directly visible to software.) Much of the data associated with a vertex is passed indirectly via a VUE handle. The information provided in vertex packets includes:

- The **URB Handle** of the VUE: This is used by the FF unit to refer to the VUE and perform any required operations on it (e.g., cause it to be read into the thread payload, dereference it, etc.).
- **Primitive Topology Information:** This information is used to identify/delineate primitive topologies in the 3D pipeline. Initially, the VF unit supplies this information, which then passes through the VS stage unchanged. GS and CLIP threads must supply this information with each vertex they produce (via the URB_WRITE message). If a FF unit directly outputs vertices (that were not generated by a thread they spawned), that FF unit is responsible for providing this information.
  - **PrimType:** The type of topology, as defined by the corresponding field of the 3DPRIMITIVE command.
  - **StartPrim:** TRUE only for the first vertex of a topology.
  - **EndPrim:** TRUE only for the last vertex of a topology.
-
- (Possibly, depending on FF unit) Data read back from the **Vertex Header** of the VUE.

Vertex URB Entry (VUE) Formats

In general, vertex data is stored in Vertex URB Entries (VUEs) in the URB, processed by CLIP threads, and only referenced by the pipeline stages indirectly via VUE handles. Therefore (for the most part) the contents/format of the vertex data is not exposed to 3D pipeline hardware – the FF units are typically only aware of the handles and sizes of VUEs.

VUEs are written in two ways:

- At the top of the 3D Geometry pipeline, the VF's InputAssembly function creates VUEs and initializes them from data extracted from Vertex Buffers as well as internally-generated data.
- VS, GS, and CLIP threads can compute, format, and write new VUEs as thread output.

There are only two points in the 3D FF pipeline where the FF units are exposed to the VUE data. Otherwise the VUE remains opaque to the 3D pipeline hardware.

- Just prior to the CLIP stage, all VUEs are read-back: Optional readback of ClipDistance values (up to 8 floats in an aligned 256-bit URB row).
- Just after the CLIP stage, on clip-generated VUEs are read-back: Readback of the Vertex Header (first 256 bits of the VUE).

Software must ensure that any VUEs subject to readback by the 3D pipeline start with a valid Vertex Header. This extends to all VUEs with the following exceptions:

- If the VS function is enabled, the VF-written VUEs are not required to have Vertex Headers, as the VS-incoming vertices are guaranteed to be consumed by the VS (i.e., the VS thread is responsible for overwriting the input vertex data).
- If the GS FF is enabled, neither VF-written VUEs nor VS thread-generated VUEs are required to have Vertex Headers, as the GS will consume all incoming vertices.
- (There is a pathological case where the CLIP state can be programmed to guarantee that all CLIP-incoming vertices are consumed – regardless of the data read back prior to the CLIP stage – and therefore only the CLIP thread-generated vertices would require Vertex Headers.)

The following table defines the Vertex Header. The Position fields are described in further detail below.

VUE Vertex Header

DWord	Bits	Description
D0	31:0	Reserved: MBZ
D1	31:0	**Render Target Array Index (RTAIndex).** This value is (eventually) used to index into a specific element of an "array" Render Target. It is read back by the GS unit (for all exiting vertices) and the Clip unit (for all clip-generated vertices), subsequently routed into the PS thread payload, and eventually included in the RTWrite DataPort message header for use by the DataPort shared function. Software is responsible for ensuring this field is zero whenever a programmable index value is not required. When a programmable index value is required, software must ensure that the correct 11-bit value is written to this field. Specifically, the kernels must perform a reange check of computed index values against [0,2047], and output zero if that range is exceeded. Note that the unmodified "renderTargetArrayIndex" must be maintained in the VUE outside of the Vertex Header. Software can force an RTAIndex of 0 to be used (effectively ignoring the setting of this DWord) by use of the **ForceZeroRTAIndex** bit (3DSTATE_CLIP). Otherwise the read-back value will be used to select an RTArray element, after being clamped to the RTArray surface's [**MinimumArrayElement, Depth**] range (SURFACE_STATE). Format: 0-based U32 index value
D2	31:0	**Viewport Index.** This value is used to select one of a possible 16 sets of viewport (VP) state parameters in the Clip unit's VertexClipTest function and in the SF unit's ViewportMapping and Scissor functions. The GS unit (even if disabled) will read back this value for all vertices exiting the GS stage and entering the Clip stage. When enabled, the GS unit will range-check the value against [0,**Maximum VPIndex**] (see GS_STATE, CLIP_STATE). After this range-check the values are sent down the pipeline and used in the Clip unit's VertexClipTest function. For vertices passing through the Clip stage, these values will also be sent to the SF unit for use in ViewportMapping and Scissor functions. The Clip unit (if enabled) will read back this value only for vertices generated by CLIP threads. The Clip unit will perform a range clamp similar to the GS unit. Software can force a value of 0 to be used by programming **Maximum VPIndex** to 0.

DWord	Bits	Description
		Format: 0-based U32 index value
D3	31:0	**Point Width.** This field specifies the width of POINT objects in screen-space pixels. It is used only for vertices within POINTLIST and POINTLIST_BF primitive topologies, and is ignored for vertices associated with other primitive topologies. This field is read back by both the GS and Clip units. Format: FLOAT32
D4	31:0	**Vertex Position X Coordinate.** This field contains the X component of the vertex's 4D space position. Format: FLOAT32
D5	31:0	**Vertex Position Y Coordinate.** This field contains the Y component of the vertex's 4D space position. Format: FLOAT32
D6	31:0	**Vertex Position Z Coordinate.** This field contains the Z component of the vertex's NDC space position. Format: FLOAT32
D7	31:0	**Vertex Position W Coordinate.** This field contains the Z component of the vertex's 4D space position. Format: FLOAT32
D8	31:0	**ClipDistance 0 Value (optional).** If the **UserClipDistance Clip Test Enable Bitmask** bit (3DSTATE_CLIP) is set, this value will be read from the URB in the Clip stage. If the value is found to be less than 0 or a NaN, the vertex's UCF<0> bit will set in the Clip unit's VertexClipTest function. If the **UserClipDistance Clip Test Enable Bitmask** bit is clear, this value will not be read back, and the vertex's UCF<0> bit will be zero by definition. Format: FLOAT32 ClipDistance Values are enabled for clip/cull test in the Clip stage in one of two modes: Normally the corresponding Enable Bitmasks are obtained from the state programmed in the last "vertex-producing" stage (VS/DS/GS) that is enabled prior to the Clip stage. E.g., if VS and DS are enabled but GS is disabled, the masks are obtained from 3DSTATE_DS. Alternatively, the Enable Bitmasks can be obtained directly from corresponding masks programmed via 3DSTATE_CLIP, through use of 3DSTATE_CLIP's Force User Clip Distance [Cull/Clip] Test Enable Bitmask state bits (see description of 3DSTATE_CLIP).
D9	31:0	**ClipDistance 1 Value (optional).** See above.

DWord	Bits	Description
D10	31:0	**ClipDistance 2 Value (optional).** See above.
D11	31:0	**ClipDistance 3 Value (optional).** See above.
D12	31:0	**ClipDistance 4 Value (optional).** See above.
D13	31:0	**ClipDistance 5 Value (optional).** See above.
D14	31:0	**ClipDistance 6 Value (optional).** See above.
D15	31:0	**ClipDistance 7 Value (optional).** See above.
	31:0	**(Remainder of Vertex Elements).** The absolute maximum size limit on this data is specified via a maximum limit on the amount of data that can be read from a VUE (including the Vertex Header) (**Vertex Entry URB Read Length** has a maximum value of 63 256-bit units). Therefore the Remainder of Vertex Elements has an absolute maximum size of 62 256-bit units. Of course the actual allocated size of the VUE can and will limit the amount of data in a VUE.

Vertex Positions

(For brevity, the following discussion uses the term map as a shorthand for "compute screen space coordinate via perspective divide followed by viewport transform".)

The "Position" fields of the Vertex Header are the only vertex position coordinates exposed to the 3D Pipeline. The CLIP and SF units are the only FF units which perform operations using these positions. The VUE will likely contain other position attributes for the vertex outside of the Vertex Header, though this information is not directly exposed to the FF units. For example, the Clip Space position will likely be required in the VUE (outside of the Vertex Header) to perform correct and robust 3D Clipping in the CLIP thread.

CLIP unit uses the **3DSTATE_CLIP.PerspectiveDivideDisable** bit to deteremine whether to perform a perspective projection (divide by w) of the read-back 4D Position.

When Perspective Divide is enabled, the Clip Space position is defined in a homogeneous 4D coordinate space (pre-perspective divide), where the visible "view volume" is defined by the APIs. The API's VS, GS or DS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a "view transform" in this computation path). When Perspective Divide is enabled, the 3D FF pipeline will perform a perspective projection (division of x,y,z by w), perform clip-test on the resulting NDC (Normalized Device Coordinates), and eventually perform viewport mapping (in the SF unit) to yield screen-space (pixel) coordinates.

When Perspective Divide is disabled, the read-back Position does not undergo perspective projection by the 3D FF pipeline.

Clip Space Position

The *clip-space* position of a vertex is defined in a homogeneous 4D coordinate space where, after perspective projection (division by W), the visible "view volume" is some canonical (3D) cuboid. Typically the X/Y extents of this cuboid are [-1,+1], while the Z extents are either [-1,+1] or [0,+1]. The API's VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a "view transform" in this computation path).

Note that, under typical perspective projections, the clip-space W coordinate is equal to the view-space Z coordinate.

A vertex's clip-space coordinates must be maintained in the VUE up to 3D clipping, as this clipping is performed in clip space.

In , vertex clip-space positions must be included in the Vertex Header, so that they can be read-back (prior to Clipping) and then subjected to perspective projection (in hardware) and subsequent use by the FF pipeline.

NDC Space Position

A perspective divide operation performed on a clip-space position yields a [X,Y,Z,RHW] NDC (Normalized Device Coordinates) space position. Here "normalized" means that visible geometry is located within the [-1,+1] or [0,+1] extent view volume cuboid (see clip-space above).

- The NDC X,Y,Z coordinates are the clip-space X,Y,Z coordinates (respectively) divided by the clip-space W coordinate (or, more correctly, the clip-space X,Y,Z coordinates are multiplied by the reciprocal of the clip space W coordinate).
    - Note that the X,Y,Z coordinates may contain INFINITY or NaN values (see below).
- The NDC RHW coordinate is the reciprocal of the clip-space W coordinate and therefore, under normal perspective projections, it is the reciprocal of the view-space Z coordinate. Note that NDC space is really a 3D coordinate space, where this RHW coordinate is retained in order to perform perspective-correct interpolation, etal. Note that, under typical perspective projections.
    - Note that the RHW coordinate make contain an INFINITY or NaN value (see below).

Screen-Space Position

Screen-space coordinates are defined as:

- X,Y coordinates are in absolute screen space (pixel coordinates, upper left origin). See Vertex X,Y Clamping and Quantization in the SF section for a discussion of the limitations/restrictions placed on screenspace X,Y coordinates.

- Z coordinate has been mapped into the range used for DepthTest.
- RHW coordinate is actually the reciprocal of clip-space W coordinate (typically the reciprocal of the view-space Z coordinate).

Vertex Fetch (VF) Stage

The Vertex Fetch Stage performs one major function: executing 3DPRIMITIVE commands. This is handled by the VF's InputAssembly function.

State

This section contains various state registers.

Control State

Register
3DSTATE_VF
3DSTATE_VF_TOPOLOGY

Index Buffer (IB) State

The 3DSTATE_INDEX_BUFFER command is used to define an *Index Buffer* (IB) used in subsequent 3DPRIMITIVE commands.

The RANDOM access mode of the 3DPRIMITIVE command involves the use of a memory-resident IB. The IB, defined via the 3DSTATE_INDEX_BUFFER command described below, contains a 1D array of 8, 16 or 32-bit index values. These index values will be fetched by the InputAssembly function, and subsequently used to compute locations in VERTEXDATA buffers from which the actual vertex data is to be fetched. (This is opposed to the SEQUENTIAL access mode were the vertex data is simply fetched sequentially from the buffers).

The following table lists which primitive topology types support the presence of Cut Indices.

Definition	Cut Index?
3DPRIM_POINTLIST	Y
3DPRIM_LINELIST	Y
3DPRIM_LINESTRIP	Y
3DPRIM_TRILIST	Y
3DPRIM_TRISTRIP	Y
3DPRIM_LINELIST_ADJ	Y
3DPRIM_LINESTRIP_ADJ	Y
3DPRIM_TRILIST_ADJ	Y
3DPRIM_TRISTRIP_ADJ	Y
3DPRIM_TRISTRIP_REVERSE	Y

Definition	Cut Index?
3DPRIM_RECTLIST	N
3DPRIM_POINTLIST_BF	Y
3DPRIM_LINESTRIP_CONT	Y
3DPRIM_LINESTRIP_BF	Y
3DPRIM_LINESTRIP_CONT_BF	Y
3DPRIM_TRIFAN_NOSTIPPLE	N

3DSTATE_INDEX_BUFFER

Vertex Buffers (VB) State

The 3DSTATE_VERTEX_BUFFERS and 3DSTATE_VF_INSTANCING commands are used to define *Vertex Buffers* (VBs) used in subsequent 3DPRIMITIVE commands.

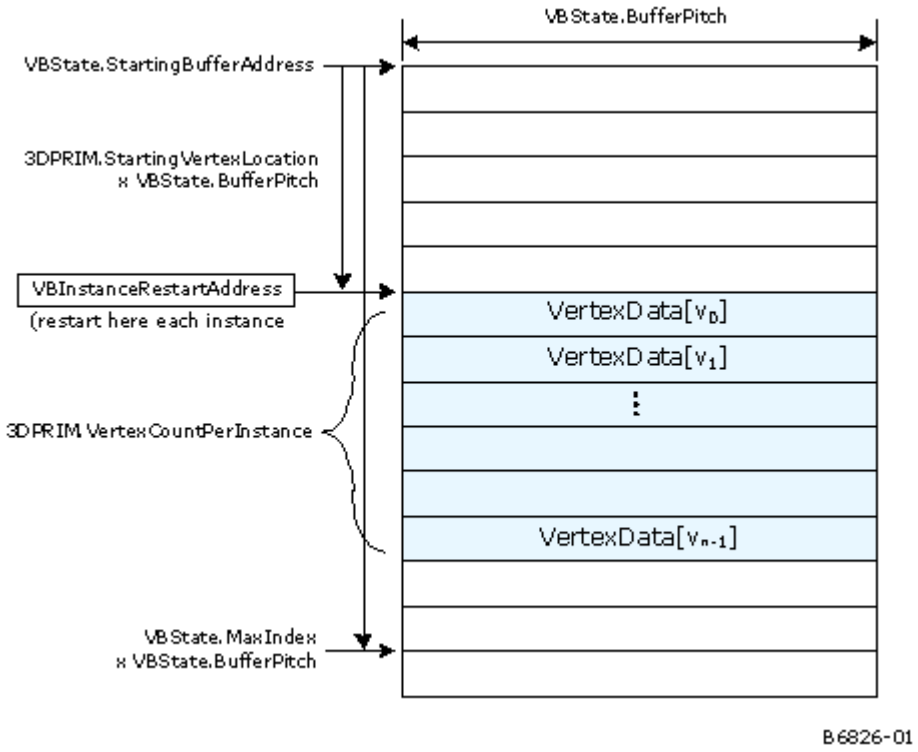
Most input vertex data is sourced from memory-resident VBs. A VB is a 1D array of structures, where the size of the structure as defined by the VB's **BufferPitch**. VBs are accessed either as *VERTEXDATA buffers* or *INSTANCEDATA buffers*, as defined by the **InstancingEnable** state in 3DSTATE_VF_INSTANCING. The VB's access type will determine whether the VF-computed VertexIndex or InstanceIndex is used to access data in the VB.

Given that the RANDOM access mode of the 3DPRIMITIVE command utilizes an IB (possibly provided by an application) to compute VB index values, VB definitions contain a **MaxIndex** value used to detect accesses beyond the end of the VBs. Any access outside the extent of a VB returns 0.

Register
3DSTATE_VERTEX_BUFFERS
VERTEX_BUFFER_STATE

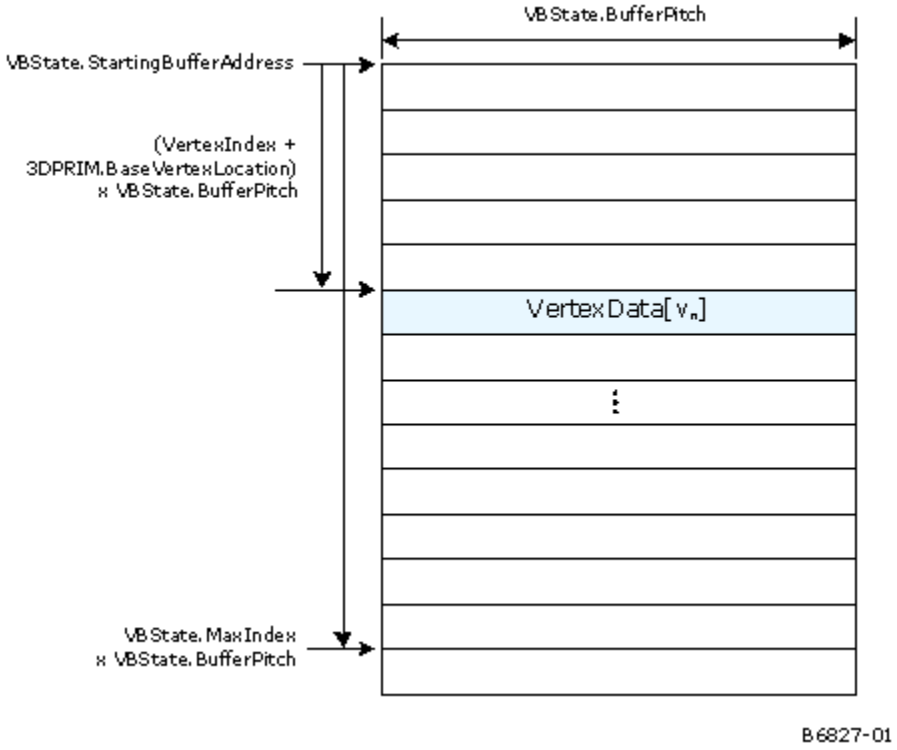
VERTEXDATA Buffers – SEQUENTIAL Access

Description
This section pertains to (a) 3DPRIMITIVE commands with VertexAccessType = SEQUENTIAL and (b) vertex elements with InstancingEnable set to DISABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



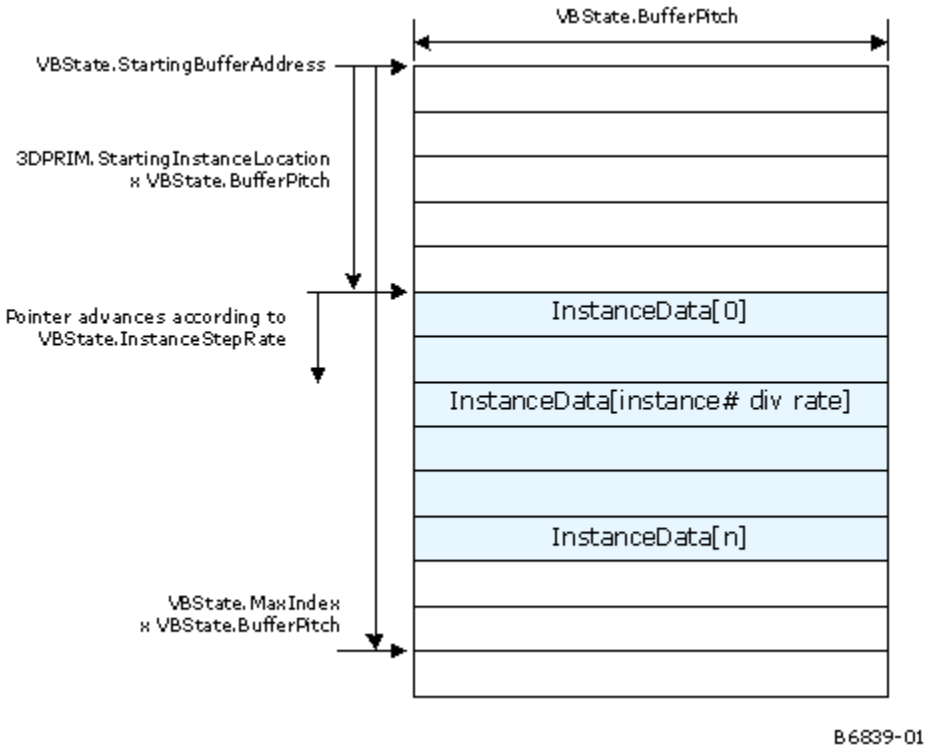
VERTEXDATA Buffers – RANDOM Access

Description
This section pertains to (a) 3DPRIMITIVE commands with VertexAccessType = RANDOM and (b) vertex elements with InstancingEnable set to DISABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



INSTANCEDATA Buffers

Description
This section pertains to vertex elements with InstancingEnable set to ENABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



Vertex Definition State

The following subsections define the state information for vertex data and describe some related processing.

Input Vertex Definition

The 3DSTATE_VERTEX_ELEMENTS command is used to define the source and format of input vertex data and the format of how it is stored in the destination VUE as part of 3DPRIMITIVE processing in the VF unit.

Two additional commands are added. 3DSTATE_VF_INSTANCING specifies the InstanceStepRate on a per-vertex-element basis. 3DSTATE_VF_SGVS specifies optional insertion of VertexID and/or InstanceID into the input vertex data (logically following the processing of the VERTEX_ELEMENT_STATE structures).

Refer to *3DPRIMITIVE Processing* below for the general flow of how input vertices are input and stored during processing of the 3DPRIMITIVE command.

Register
VERTEX_ELEMENT_STATE
3DSTATE_VERTEX_ELEMENTS
3D_Vertex_Component_Control
3DSTATE_VF_INSTANCING
3DSTATE_VF_SGVS
3DSTATE_VF_COMPONENT_PACKING

3D Primitive Command

Following are 3D Primitive Commands:

3DPRIMITIVE

3D Primitive Topology Type Encoding

The following table defines the encoding of the Primitive Topology Type field. See *3D Pipeline* for details, programming restrictions, diagrams, and a discussion of the basic primitive types.

3D_Prim_Topo_Type

Functions

This section covers the various functions for Vertex Fetch.

Input Assembly

The VF's InputAssembly function includes (for each vertex generated):

- Generation of VertexIndex and InstanceIndex for each vertex, possibly via use of an Index Buffer.
- Lookup of the VertexIndex in the Vertex Cache (if enabled)
- If a cache miss is detected:
  - Use of computed indices to fetch data from memory-resident vertex buffers
  - Format conversion of the fetched vertex data
  - Assembly of the format conversion results (and possibly some internally generated data) to form the complete "input" (raw) vertex
  - Storing the input vertex data in a Vertex URB Entry (VUE) in the URB
  - Output of the VUE handle of the input vertex to the VS stage
- If a cache hit is detected, the VUE handle from the Vertex Cache is passed to the VS stage (marked as a cache hit to prevent any VS processing).

Vertex Assembly

The VF utilizes a number of VERTEX_ELEMENT state structures to define the contents and format of the vertex data to be stored in Vertex URB Entries (VUEs) in the URB. See below for a detailed description of the command used to define these structures (3DSTATE_VERTEX_ELEMENTS).

Each active VERTEX_ELEMENT structure defines up to 4 contiguous DWords of VUE data, where each DWord is considered a "component" of the vertex element. The starting destination DWord offset of the vertex element in the VUE is specified, and the VERTEX_ELEMENT structures must be defined with monotonically increasing VUE offsets. For each component, the source of the component is specified. The source may be a constant (0, 0x1, or 1.0f), a generated ID (VertexID, InstanceID or PrimitiveID), or a component of a structure in memory (e.g,. the Y component of an XYZW position in memory). In the case

of a memory source, the Vertex Buffer sourcing the data, and the location and format of the source data with that VB are specified.

The VF's Vertex Assembly process can be envisioned as the VF unit stepping through the VERTEX_ELEMENT structures in order, fetching and format-converting the source information (if memory resident), and storing the results in the destination VUE.

The information supplied via the 3DSTATE_VF_SGVS command is also used to optionally insert VertexID and/or InstanceID into the input vertex data, after the VERTEX_ELEMENT structures are processed.

Vertex Cache

The VF stage communicates with the VS stage in order to implement a Vertex Cache function in the 3D pipeline. The Vertex Cache is strictly a performance-enhancing feature and has no impact on 3D pipeline results (other than a few statistics counters).

The Vertex Cache contains the VUE handles of VS-output (shaded) vertices if the VS function is enabled, and the VUE handles of VF-output (raw) vertices if the VS function is disabled. (Note that the actual vertex data is held in the URB, and only the handles of the vertices are stored in the cache). In either case, the contents of the cache (VUE handles) are tagged with the VertexIndex value used to fetch the input vertex data. The rationale for using the VertexIndex as the tag is that (assuming no other state or parameters change) a vertex with the same VertexIndex as a previous vertex will have the same input data, and therefore the same result from the VF+VS function.

Note that any change to the state controlling the InputAssembly function (e.g., vertex buffer definition), or any change to the state controlling the VS function (if enabled) (e.g., VS kernel), will result in the Vertex Cache being invalidated. In addition, any non-trivial use of instancing (i.e., more than one instance per 3DPRIMITIVE command and the inclusion of instance data in the input vertex) will effectively invalidate the cache between instances, as the InstanceIndex is not included in the cache tag. See Vertex Caching in *Vertex Shader* for more information on the Vertex Cache (e.g., when it is implicitly disabled, etc.)

Modern 3D APIs include the notion of native support for *instanced geometry*. The premise is that the application models a high-level object, and then proceeds to process (render) multiple instances of the object, with each instance including some modification to the data associated with the object. For example, a single chair object could be modeled and rendered multiple times, with each instance using a different position transformation (to place several chairs around a table). That transformation matrix would be considered *instance data*, in that it would be fixed for each instance of the object, changing only between instances. This is opposed to the *vertex data*, which could be unique for each vertex of the object. Note that the instance data is replicated at each vertex as part of the Input Assembly process, and so could also be considered "vertex data" in that respect. In fact, the notion of instance data is confined to the VF stage, and the remainder of the 3D pipeline is unaware of the distinction.

Although the ability to perform instanced geometry ("manually") was available in legacy APIs, modern APIs add support for a single Draw() call to process multiple instances. The application specifies the number of instances, and the number of vertices in the instanced object (instance) in addition to other

parameters discussed later. The Draw() support effectively becomes a nested loop, with the outer loop dealing with instance variables, and the inner loop sequencing through the object vertices.

Given the flexibility and programmability of the 3D pipeline, the division between vertex and instance data is arbitrary. The application simply marks input data buffers as either containing vertex or instance data. In the outer instance loop, the data specified as coming from *instance buffers* is (effectively) fetched/generated and then associated with each vertex of the instance in the inner loop and combined with the vertex data. New instance data is then gathered for the next instance (the "stepping" of instance data is specified on a per-buffer basis, and different portions of the instance data are allowed to step at different rates -- refer to details below).

The hardware interface to supply instancing state information is slightly different. Individual vertex elements (instead of buffers) are tagged as instanced or not.

Input Data: Push Model vs. Pull Model

Given the programmability of the pipeline, and the ability of shaders to input (load/sample) data from memory buffers in an arbitrary fashion, the decision arises in whether to push instance/vertex data into the front of the pipeline or defer the data access (pull) to the shaders that require it. Modern APIs directly support the latter model via *auto-generated IDs* in the Input Assembly function. An incrementing *VertexID*, *InstanceID*, and *PrimitiveID* are generated in the Input Assembly process, and these values can be declared as input to the "first enabled, relevant" shader. That shader can, for example, use the HW-generated ID as an index into a memory resource such as a constant buffer or vertex buffer. The 3D pipeline HW supports these IDs as required by the APIs.

There are tradeoffs involved in deciding between these models. For vertex data, it is probably always better to push the data into the pipeline, as the VF hardware attempts to cover the latency of the data fetch. The decision is less clear for instance data, as pushing instance data leads to larger Vertex URB entries which will be holding redundant data (as the instance data for vertices of an object are by definition the same). Regardless, the GEN 3D pipeline supports both models.

Generated IDs

Note that the generated IDs are considered separate from any offset computations performed by the VF unit, and are therefore described separately here.

The VF generates InstanceID, VertexID, and PrimitiveID values as part of the InputAssembly process.

VertexID and InstanceID are only allowed to be inserted into the input vertex data as it is gathered and written into the URB as a VUE.

The definition/use of PrimitiveID is more complicated than the other auto-generated IDs. PrimitiveID is associated with an "object" and not a particular vertex.

Description
It is only available to the GS and HS as a special non-vertex input and the PS as a constant-interpolated attribute. It is not seen by the VS or DS at all.

The PrimitiveID therefore is kept separate from the vertex data. Take for example a TRILIST primitive topology: It should be possible to share vertices between triangles in the list (i.e., reuse the VS output of a vertex), even though each triangle has a different PrimitiveID associated with it.

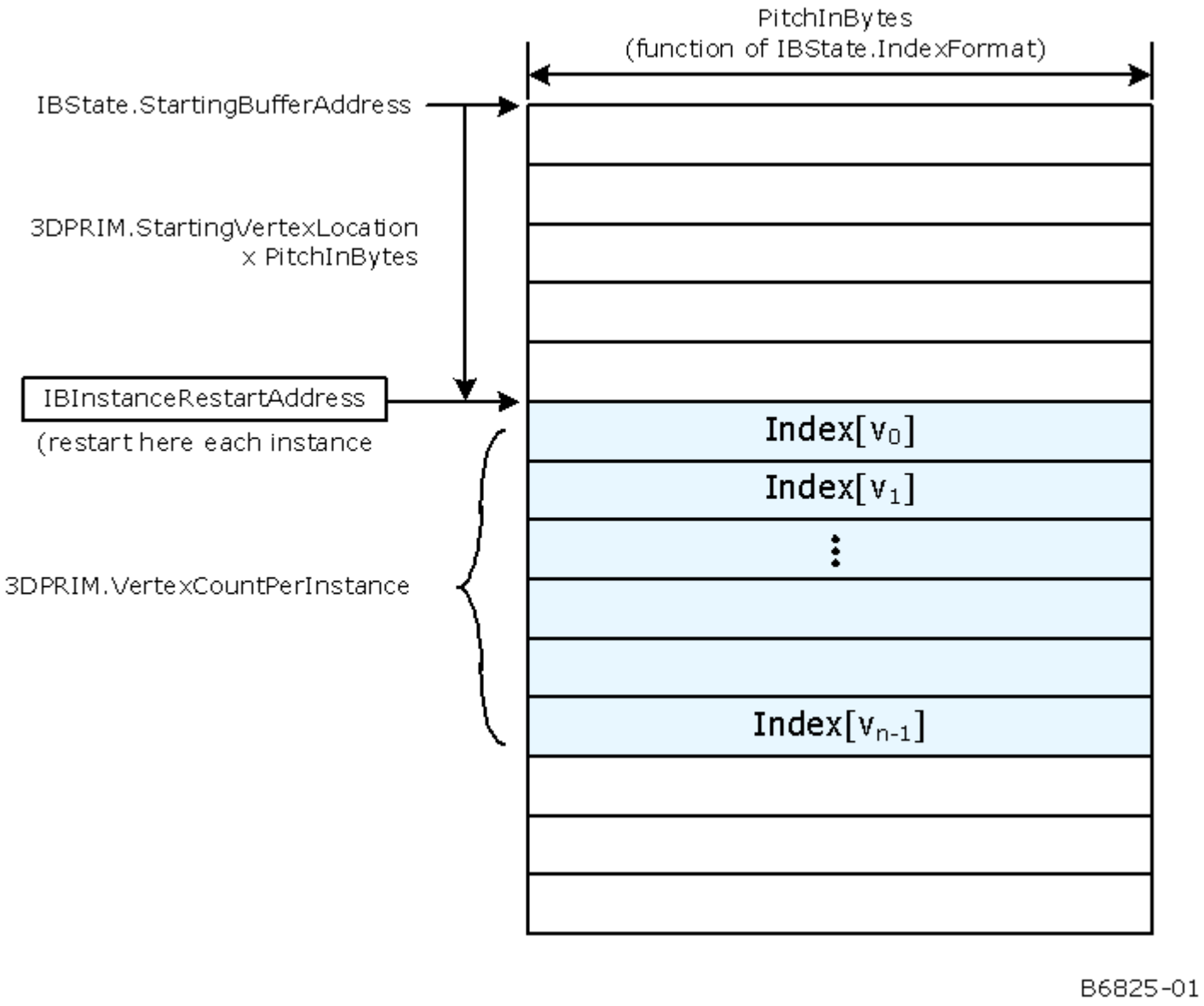
The optional insertion of VertexID and/or InstanceID into the input vertex data occurs as a separate step after the processing of VERTEX_ELEMENT structures, and is controlled via the 3DSTATE_VF_SGVS command.

PrimitiveID is generated by hardware, plumbed down into the HS, GS and SF stages. It is passed along in HS/GS thread payloads. Software can also select PrimitiveID to be swizzled into vertex attribute data in the SF stage, though only if neither the HS nor GS stages are enabled.

3D Primitive Processing

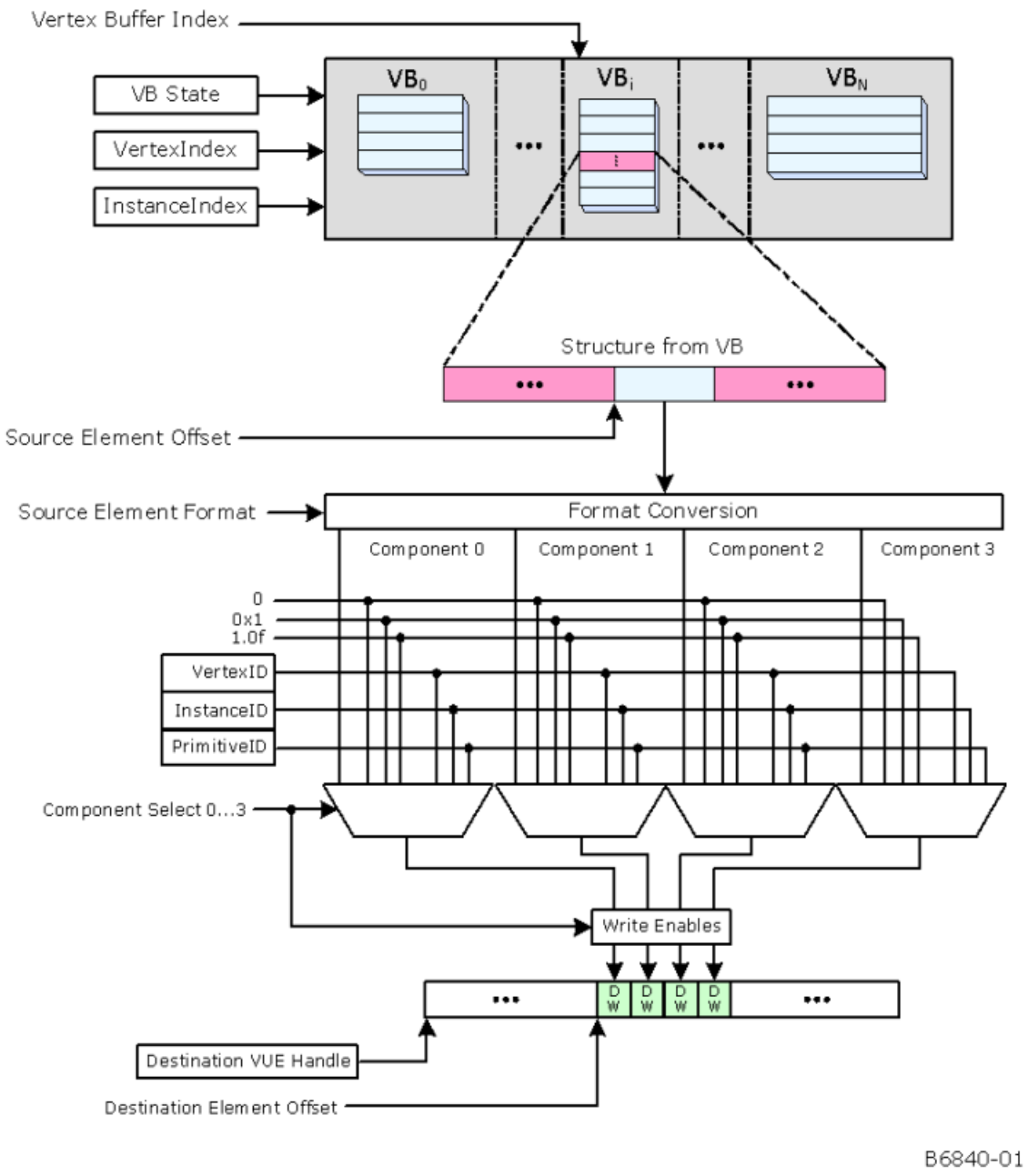
Index Buffer Access

The following figure illustrates how the Index Buffer is accessed.



Vertex Element Data Path

The following diagram shows the path by which a vertex element within the destination VUE is generated and how the fields of the VERTEX_ELEMENT_STATE structure is used to control the generation.



FormatConversion

Once the VE source data has been fetched, it is subjected to format conversion. The output of format conversion is up to 4 32-bit components, each either integer or floating-point (as specified by the **Source Element Format**). See *Sampler* for conversion algorithms.

The following table lists the valid **Source Element Format** selections, along with the format and availability of the converted components (if a component is listed as -, it cannot be used as the source of a VUE component). **Note:** This table is a subset of the list of supported surface formats defined in the

Sampler chapter. Please refer to that table as the "master list". This table is here only to identify the components available (per format) and their format.

Source Element Formats Supported in VF Unit

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32G32B32A32_FLOAT	FLOAT	R	G	B	A
R32G32B32A32_SINT	SINT	R	G	B	A
R32G32B32A32_UINT	UINT	R	G	B	A
R32G32B32A32_UNORM	FLOAT	R	G	B	A
R32G32B32A32_SNORM	FLOAT	R	G	B	A
R64G64_FLOAT	FLOAT	R	G	-	-
R32G32B32A32_SSCALED	FLOAT	R	G	B	A
R32G32B32A32_USCALED	FLOAT	R	G	B	A
R32G32B32A32_SFIXED	FLOAT	R	G	B	A
R64G64_PASSTHRU	NONE	R	G	-	-
R32G32B32_FLOAT	FLOAT	R	G	B	-
R32G32B32_SINT	SINT	R	G	B	-
R32G32B32_UINT	UINT	R	G	B	-
R32G32B32_UNORM	FLOAT	R	G	B	-
R32G32B32_SNORM	FLOAT	R	G	B	-
R32G32B32_SSCALED	FLOAT	R	G	B	-
R32G32B32_USCALED	FLOAT	R	G	B	-
R32G32B32_SFIXED	FLOAT	R	G	B	-
R16G16B16A16_UNORM	FLOAT	R	G	B	A
R16G16B16A16_SNORM	FLOAT	R	G	B	A
R16G16B16A16_SINT	SINT	R	G	B	A
R16G16B16A16_UINT	UINT	R	G	B	A
R16G16B16A16_FLOAT	FLOAT	R	G	B	A
R32G32_FLOAT	FLOAT	R	G	-	-
R32G32_SINT	SINT	R	G	-	-
R32G32_UINT	UINT	R	G	-	-
R32G32_UNORM	FLOAT	R	G	-	-
R32G32_SNORM	FLOAT	R	G	-	-
R64_FLOAT	FLOAT	R	-	-	-
R16G16B16A16_SSCALED	FLOAT	R	G	B	A
R16G16B16A16_USCALED	FLOAT	R	G	B	A

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32G32_SSCALED	FLOAT	R	G	-	-
R32G32_USCALED	FLOAT	R	G	-	-
R32G32_SFIXED	FLOAT	R	G	-	-
R64_PASSTHRU	NONE	R	-	-	-
B8G8R8A8_UNORM	FLOAT	B	G	R	A
R10G10B10A2_UNORM	FLOAT	R	G	B	A
R10G10B10A2_UINT	UINT	R	G	B	A
R10G10B10_SNORM_A2_UNORM	FLOAT	R	G	B	A
R8G8B8A8_UNORM	FLOAT	R	G	B	A
R8G8B8A8_SNORM	FLOAT	R	G	B	A
R8G8B8A8_SINT	SINT	R	G	B	A
R8G8B8A8_UINT	UINT	R	G	B	A
R16G16_UNORM	FLOAT	R	G	-	-
R16G16_SNORM	FLOAT	R	G	-	-
R16G16_SINT	SINT	R	G	-	-
R16G16_UINT	UINT	R	G	-	-
R16G16_FLOAT	FLOAT	R	G	-	-
B10G10R10A2_UNORM	FLOAT	R	G	B	A
R11G11B10_FLOAT	FLOAT	R	G	B	-
R32_SINT	SINT	R	-	-	-
R32_UINT	UINT	R	-	-	-
R32_FLOAT	FLOAT	R	-	-	-
R32_UNORM	FLOAT	R	-	-	-
R32_SNORM	FLOAT	R	-	-	-
R10G10B10X2_USCALED	FLOAT	R	G	B	-
R8G8B8A8_SSCALED	FLOAT	R	G	B	A
R8G8B8A8_USCALED	FLOAT	R	G	B	A
R16G16_SSCALED	FLOAT	R	G	-	-
R16G16_USCALED	FLOAT	R	G	-	-
R32_SSCALED	FLOAT	R	-	-	-
R32_USCALED	FLOAT	R	-	-	-
R8G8_UNORM	FLOAT	R	G	-	-
R8G8_SNORM	FLOAT	R	G	-	-
R8G8_SINT	SINT	R	G	-	-
R8G8_UINT	UINT	R	G	-	-

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R16_UNORM	FLOAT	R	-	-	-
R16_SNORM	FLOAT	R	-	-	-
R16_SINT	SINT	R	-	-	-
R16_UINT	UINT	R	-	-	-
R16_FLOAT	FLOAT	R	-	-	-
R8G8_SSCALED	FLOAT	R	G	-	-
R8G8_USCALED	FLOAT	R	G	-	-
R16_SSCALED	FLOAT	R	-	-	-
R16_USCALED	FLOAT	R	-	-	-
R8_UNORM	FLOAT	R	-	-	-
R8_SNORM	FLOAT	R	-	-	-
R8_SINT	SINT	R	-	-	-
R8_UINT	UINT	R	-	-	-
R8_SSCALED	FLOAT	R	-	-	-
R8_USCALED	FLOAT	R	-	-	-
R8G8B8_UNORM	FLOAT	R	G	B	-
R8G8B8_SNORM	FLOAT	R	G	B	-
R8G8B8_SSCALED	FLOAT	R	G	B	-
R8G8B8_USCALED	FLOAT	R	G	B	-
R8G8B8_SINT	SINT	R	G	B	-
R8G8B8_UINT	UINT	R	G	B	-
R64G64B64A64_FLOAT	FLOAT	R	G	B	A
R64G64B64_FLOAT	FLOAT	R	G	B	A
R16G16B16_FLOAT	FLOAT	R	G	B	-
R16G16B16_UNORM	FLOAT	R	G	B	-
R16G16B16_SNORM	FLOAT	R	G	B	-
R16G16B16_SSCALED	FLOAT	R	G	B	-
R16G16B16_USCALED	FLOAT	R	G	B	-
R16G16B16_UINT	UINT	R	G	B	-
R16G16B16_SINT	SINT	R	G	B	-
R32_SFIXED	FLOAT	R	-	-	-
R10G10B10A2_SNORM	FLOAT	R	G	B	A
R10G10B10A2_USCALED	FLOAT	R	G	B	A
R10G10B10A2_SSCALED	FLOAT	R	G	B	A
R10G10B10A2_SINT	SINT	R	G	B	A

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
B10G10R10A2_SNORM	FLOAT	R	G	B	A
B10G10R10A2_USCALED	FLOAT	R	G	B	A
B10G10R10A2_SSCALED	FLOAT	R	G	B	A
B10G10R10A2_UINT	UINT	R	G	B	A
B10G10R10A2_SINT	SINT	R	G	B	A
R64G64B64A64_PASSTHRU	NONE	R	G	B	A
R64G64B64_PASSTHRU	NONE	R	G	B	-

DestinationFormatSelection

The **Component Select 0..3** bits are then used to select, on a per-component basis, which destination components will be written and with which value. The supported selections are the converted source component, VertexID, InstanceID, PrimitiveID, the constants 0 or 1.0f, or nothing (VFCOMP_NO_STORE). If a converted component is listed as '-' (not available) in Source Element Formats supported in VF Unit. It must not be selected (via VFCOMP_STORE_SRC), or an UNPREDICTABLE value will be stored in the destination component.

The selection process sequences from component 0 to 3. Once a **Component Select** of VFCOMP_NO_STORE is encountered, all higher-numbered **Component Select** settings must also be programmed as VFCOMP_NO_STORE. It is therefore not permitted to have 'holes' in the destination VE.

Dangling Vertex Removal

The last functional stage of processing of the 3DPRIMITIVE command is the removal of "dangling" vertices. This stage includes the discarding of primitive topologies without enough vertices for a single object (e.g., a TRISTRIP with only two vertices), as well as the discarding of trailing vertices that do not form a complete primitive (e.g., the last two vertices of a 5-vertex TRILIST). 3D APIs typically require these vertices to be (effectively) discarded before the VS stage.

Statistics Gathering

This function is best described as a filter operating on the vertex stream emitted from the processing of the 3DPRIMITIVE. The filter inputs the PrimType, PrimStart, and PrimEnd values associated with the generated vertices. The filter only outputs primitive topologies without dangling vertices. This requires the filter to (a) be able to buffer some number of vertices, and (b) be able to remove dangling vertices from the pipeline and dereference the associated VUE handles.

3DSTATE_VF_STATISTICS

Vertices Generated

VF will increment the IA_VERTICES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) for each vertex it fetches, even if that vertex comes from a cache rather than directly from a vertex buffer

in memory. Any “dangling” vertices (fetched vertices that are part of an incomplete object) will not be included.

Objects Generated

VF will increment the IA_PRIMITIVES_COUNT Register (see Memory Interface Registers in vol1a System Overview) for each object (point, line, triangle, or quadrilateral) that it forwards down the pipeline.

Note
For LINELOOP, the last (closing) line object is counted.

Vertex Shader (VS) Stage

The Vertex Shader (VS) stage of the 3D Pipeline is used to perform processing (“shading”) of vertices after they are assembled and written to the URB by the VF function. The primary function of the VS stage is to pass vertices that miss in the VS Cache to VS threads, and then pass the VS thread-generated vertices down the pipeline. Vertices that hit in the VS Cache have already been shaded and are therefore passed down the pipeline unmodified.

When the VS stage is disabled, vertices flow through the unit unmodified (i.e., as written by the VF unit).

State

Register
3DSTATE_VS
3DSTATE_CONSTANT_VS
3DSTATE_PUSH_CONSTANT_ALLOC_VS
3DSTATE_BINDING_TABLE_POINTERS_VS
3DSTATE_SAMPLER_STATE_POINTERS_VS
3DSTATE_URB_VS

Payloads

SIMD4x2 Payload

The following table describes the payload delivered to VS threads.

VS Thread Payload (SIMD4x2)

DWord	Bits	Description
	30:0	Reserved
R0.6	31:24	Reserved

DWord	Bits	Description
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset:** Specifies the extent of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID:** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. **Format:** Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address**or the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). (See *3D Pipeline* for further description). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:16	Reserved

DWord	Bits	Description
	15:0	**URB Return Handle 1:** This is the 64B-aligned URB offset where the EU's upper channels (DWords 7:4) results are to be stored. If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). (See *Generic FF Unit* for further description). Format: **Format** U14 64B-aligned URB offset.
R0.0	31:16	Reserved
	15:0	**URB Return Handle 0:** This is the 64B-aligned URB offset where the EU's lower channels (DWords 3:0) results are to be stored. (See *Generic FF Unit* for further description). Format: **Format** U14 64B-aligned URB offset.
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
Varies	255:0	**Vertex Data:** Data from (possibly) one or (more typically) two Vertex URB Entries is passed to the thread in the thread payload. The **Vertex URB Entry Read Offset** and **Vertex URB Entry Read Length** state variables define the regions of the URB entries that are read from the URB and passed in the thread payload. These SVs can be used to provide a subset of the URB data as required by SW. The vertex data is laid out in the thread header in an interleaved format. The lower DWords (0-3) of these GRF registers always contain data from a Vertex URB Entry. The upper DWords (4-7) may contain data from another Vertex URB Entry. This allows two vertices to be processed (shaded) in parallel SIMD8 fashion. The VS kernel is not aware of the validity of the upper vertex.

SIMD8 Payload

The following table describes the payload delivered to VS threads.

SIMD8 VS Thread Payload

DWord	Bits	Description
R0.7	31	
	30:0	Reserved

DWord	Bits	Description
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID:** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. **Format:** Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address** or **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	**Description** Reserved.
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). (See *3D Pipeline* for further description.) Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2 : R0.0	31:0	Reserved: MBZ

DWord	Bits	Description
R1.7	31:0	Vertex 7 URB Return Handle (see R1.0)
R1.6	31:0	Vertex 6 URB Return Handle (see R1.0)
R1.5	31:0	Vertex 5 URB Return Handle (see R1.0)
R1.4	31:0	Vertex 4 URB Return Handle (see R1.0)
R1.3	31:0	Vertex 3 URB Return Handle (see R1.0)
R1.2	31:0	Vertex 2 URB Return Handle (see R1.0)
R1.1	31:0	Vertex 1 URB Return Handle (see R1.0)
R1.0	31:16	Reserved
	15:0	**Vertex 0 URB Return Handle.** This is the offset within the URB where Vertex 0 is to be stored. Format: 64B-granular offset into the URB
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
		Vertex Data: Input data for the 8 input vertices is located here. Vertex0 data is passed in DW0 of these GRFs, and Vertex 7 data is passed in DW7. The first GRF contains Element 0 Component 0 for all 8 vertices, followed by components 1-3 in the three subsequent GRFs. This is followed by GRFs containing Element 1, and so on, up to the number of elements specified by Vertex URB Read Length. Note that the maximum limit is 30 elements per vertex, though the practical limiit imposed by the compiler is likely lower.
Rv.7	31:0	Vertex 7 Element 0 Component 0
Rv.6	31:0	Vertex 6 Element 0 Component 0
Rv.5	31:0	Vertex 5 Element 0 Component 0
Rv.4	31:0	Vertex 4 Element 0 Component 0
Rv.3	31:0	Vertex 3 Element 0 Component 0
Rv.2	31:0	Vertex 2 Element 0 Component 0
Rv.1	31:0	Vertex 1 Element 0 Component 0
Rv.0	31:0	Vertex 0 Element 0 Component 0
Rv+1.7	31:0	Vertex 7 Element 0 Component 1
Rv+1.6	31:0	Vertex 6 Element 0 Component 1
Rv+1.5	31:0	Vertex 5 Element 0 Component 1
Rv+1.4	31:0	Vertex 4 Element 0 Component 1
Rv+1.3	31:0	Vertex 3 Element 0 Component 1
Rv+1.2	31:0	Vertex 2 Element 0 Component 1
Rv+1.1	31:0	Vertex 1 Element 0 Component 1

DWord	Bits	Description
Rv+1.0	31:0	Vertex 0 Element 0 Component 1
..		Vertex 0-7 Element 0 Component 2,3
..		Vertex 0-7 Element 1 Component 0-3
..		Vertex 0-7 Element 2-N Component 0-3

Hull Shader (HS) Stage

The Hull Shader (HS) stage of the pipeline is used to process patchlist (PATCHLIST_*n*) topologies in support of higher-order surface (HOS) tessellaton. If the HS stage is enabled, each incoming patch object is processed by a possible series of HS threads. The combined output of these threads is a Patch URB Entry ("patch record") written to the URB. This patch record is used by subsequent stages (TE, DS) to complete the HOS tessellation operations.

The vertices associated with patchlist primitives are also referred to as "Input Control Points" (ICPs) to contrast them with any "Output Control Points" the HS threads may write to the patch record. (The definition and use of OCPs are outside the scope of this document).

The HS stage also performs statistics counting. Incomplete topologies do not reach the HS stage.

The HS, TE, and DS stages must be enabled and disabled together. When these stages are disabled, all topologies (including patchlist topologies) simply pass through to the GS stage. When these stages are enabled, only patchlist topologies should be issued to the pipeline, otherwise behavior is UNDEFINED.

Programming Note	
Context:	Hull Shader (HS) Stage
Tessellation is not supported in HPCXTs in which case the HS, TE, and DS stages must all be disabled.	

State

This section contains the state registers for the Hull Shader

Register
3DSTATE_HS
3DSTATE_PUSH_CONSTANT_ALLOC_HS
3DSTATE_CONSTANT_HS
3DSTATE_CONSTANT(Body)
3DSTATE_BINDING_TABLE_POINTERS_HS
3DSTATE_SAMPLER_STATE_POINTERS_HS
3DSTATE_URB_HS

Functions

Patch Object Staging

The HS unit accepts patchlist topologies as a stream of incoming vertices. Depending on the number of vertices per patch object (as specified by the PATCHLIST_*n* topology), the HS thread assembles each complete patch object and passes it (its vertices, PrimitiveID, etc.) to HS thread(s) as described below.

HS Thread Execution

Input to HS threads is comprised of:

- Input Control Points (incoming patch vertices), pushed into the payload and/or passed indirectly via URB handles.
- Push Constants (common to all threads)
- Patch Data handle
- Resources available via binding table entries (accessed through shared functions)
- Miscellaneous payload fields (Instance Number, etc.)

Typically the only output of the HS threads is the Patch URB Entry (patch record). All thread instances for an input patch are passed the same patch record handle. As the (possibly concurrent) threads can both read and write the patch record, it is up to the kernels to ensure deterministic results. One approach would be to use the thread's Instance Number as an index for URB write destinations.

Programming Note	
Context:	HS Thread Execution - Dual Patch Execution
• In DUAL_PATCH mode HS threads operate on up to two patches, with Patch 0 data residing in the four lower channels and Patch 1 residing in the four upper channels. (Note that, for boundary cases, Patch 1 may not be present, with the upper four channels disabled via Dispatch Mask.) Correspondingly, up to two output patch URB handles are provided. • DUAL_PATCH mode does not support thread instancing,	

HS Thread Dispatch Mask

The HS stage controls the initial value loaded into the EU's Dispatch Mask state register as part of thread dispatch.

SINGLE_PATCH Dispatch Mask

In SINGLE_PATCH mode, the EU Dispatch Mask is initialized at thread dispatch to 0x000000FF.

DUAL_PATCH Dispatch Mask

In DUAL_PATCH mode, the EU Dispatch Mask is initialized as a function of the number of patches included in the thread dispatch, as follows:

- 1 patch: 0x0000000F
- 2 patches 0x000000FF

8_PATCH Dispatch Mask

In 8_PATCH mode, the EU Dispatch Mask is initialized as a function of the number of patches included in the thread dispatch, as follows:

- 1 patch: 0x00000001
- 2 patches 0x00000003
- 3 patches: 0x00000007
- 4 patches: 0x0000000F
- 5 patches: 0x0000001F
- 6 patches: 0x0000003F
- 7 patches: 0x0000007F

8 patches: 0x000000FF

Patch URB Entry (Patch Record) Output

For each patch, the HS thread(s) generate a single patch record, starting with a fixed 32B Patch Header.

When the final thread instance terminates, the patch record handle is passed down the pipeline to the Tessellation Engine (TE).

Patch Header DW0-7

The first 8 DWords of the patch record is defined as a "Patch Header". The Patch Header is written by an HS thread and read by the TE stage. It normally contains up to six **Tessellation Factors** (TFs) that determine how finely the TE stage needs to tessellate a domain (if at all).

The following tables show the fixed layouts of the Patch Header DW0-7, depending on DomainType.

Patch Header (QUAD Domain)

DWord	Bits	Description
7	31:0	**UEQ0 Tessellation Factor** Format: FLOAT32
6	31:0	**VEQ0 Tessellation Factor** Format: FLOAT32
5	31:0	**UEQ1 Tessellation Factor** Format: FLOAT32
4	31:0	**VEQ1 Tessellation Factor** Format: FLOAT32
3	31:0	**Inside U Tessellation Factor** Format: FLOAT32
2	31:0	**Inside V Tessellation Factor** Format: FLOAT32

DWord	Bits	Description
1	31:0	Reserved : MBZ
0	31:1	Reserved : MBZ
	0	Description Reserved: MBZ

Patch Header (TRI Domain)

DWord	Bits	Description
7	31:0	**UEQ0 Tessellation Factor** Format: FLOAT32
6	31:0	**VEQ0 Tessellation Factor** Format: FLOAT32
5	31:0	**WEQ0 Tessellation Factor** Format: FLOAT32
4	31:0	**Inside Tessellation Factor** Format: FLOAT32
3-1	31:0	Reserved : MBZ
0	31:1	Reserved : MBZ
	0	Description Reserved: MBZ

Patch Header (ISOLINE Domain)

DWord	Bits	Description
7	31:0	**Line Detail Tessellation Factor** Format: FLOAT32
6	31:0	**Line Density Tessellation Factor** Format: FLOAT32
5-0	31:0	Reserved : MBZ

Statistics Gathering

HS Invocations

The HS unit controls the HS_INVOCATIONS counter, which counts the number of patches processed by the HS stage.

Payloads

SINGLE_PATCH Payload

The following table shows the layout of the payload delivered to HS threads. Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

Patch object vertex (ICP) data can be passed by value (data pushed in the payload) and/or by reference (URB handle pushed in the payload).

SINGLE_PATCH HS Thread Payload

GRF DWord	Bits	Description
R0.7	31	
	30:0	Reserved.
R0.6	31	Dereference Thread This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.
	23:0	Thread ID. This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address. Format = GeneralStateOffset[31:10]
	9.0	Reserved.
	8:0	FFTID. This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for HW Implementation Use.
R0.4	31:5	Binding Table Pointer: Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	Reserved.

GRF DWord	Bits	Description
R0.3	31:5	Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved.
	3:0	Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Programming Notes: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:24	Reserved.
	23:17	Instance Number. A patch-relative instance number between 0 and InstanceCount-1. Format = U7
	16:13	Barrier Index. This index is to be used in any BarrierMsgs sent by this thread to the Gateway. Format = U4
	12:0	Reserved.
R0.1	31:0	Primitive ID. This field contains the Primitive ID associated with the patch. Format: U32
R0.0	31:16	Reserved.
	15:0	Patch Data Record URB Return Handle. Format: **Format** U14 64B-aligned URB offset.
R1 is only included for dispatches that have Include Vertex Handles enabled.		
R1.7	31:16	Reserved.

GRF DWord	Bits	Description
	15:0	ICP 7 Handle Format: Format U14 64B-aligned URB offset.
R1.6	31:16	Reserved.
	15:0	ICP 6 Handle
R1.5	31:16	Reserved.
	15:0	ICP 5 Handle
R1.4	31:16	Reserved.
	15:0	ICP 4 Handle
R1.3	31:16	Reserved.
	15:0	ICP 3 Handle
R1.2	31:16	Reserved.
	15:0	ICP 2 Handle
R1.1	31:16	Reserved.
	15:0	ICP 1 Handle
R1.0	31:16	Reserved.
	15:0	ICP 0 Handle
R2 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >7		
R2.7	31:16	Reserved.
	15:0	ICP 15 Handle
R2.6	31:16	Reserved.
	15:0	ICP 14 Handle
R2.5	31:16	Reserved.
	15:0	ICP 13 Handle
R2.4	31:16	Reserved.
	15:0	ICP 12 Handle
R2.3	31:16	Reserved.
	15:0	ICP 11 Handle
R2.2	31:16	Reserved.
	15:0	ICP 10 Handle
R2.1	31:16	Reserved.

GRF DWord	Bits	Description
	15:0	ICP 9 Handle
R2.0	31:16	Reserved.
	15:0	ICP 8 Handle
R3 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >15		
R3.7	31:16	Reserved.
	15:0	ICP 23 Handle
R3.6	31:16	Reserved.
	15:0	ICP 22 Handle
R3.5	31:16	Reserved.
	15:0	ICP 21 Handle
R3.4	31:16	Reserved.
	15:0	ICP 20 Handle
R3.3	31:16	Reserved.
	15:0	ICP 19 Handle
R3.2	31:16	Reserved.
	15:0	ICP 18 Handle
R3.1	31:16	Reserved.
	15:0	ICP 17 Handle
R3.0	31:16	Reserved.
	15:0	ICP 16 Handle
R4 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >23		
R4.7	31:16	Reserved.
	15:0	ICP 31 Handle
R4.6	31:16	Reserved.
	15:0	ICP 30 Handle
R4.5	31:16	Reserved.
	15:0	ICP 29 Handle
R4.4	31:16	Reserved.
	15:0	ICP 28 Handle
R4.3	31:16	Reserved.
	15:0	ICP 27 Handle
R4.2	31:16	Reserved.
	15:0	ICP 26 Handle

GRF DWord	Bits	Description
R4.1	31:16	Reserved.
	15:0	ICP 25 Handle
R4.0	31:16	Reserved.
	15:0	ICP 24 Handle
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
[Varies] optional	255:0	ICP Vertex Data (optional): There can be up to 32 vertices supplied, each with a size defined by the Vertex URB Entry Read Length state. Vertex 0 DWord 0 is located at Rn.0, Vertex 0 DWord 1 is located at Rn.1, etc. Vertex 1 DWord 0 immediately follows the last DWord of Vertex 0, and so on.

DUAL_PATCH Payload

The following table shows the layout of the payload delivered to HS threads. Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

Patch object vertex (ICP) data can be passed by value (data pushed in the payload) and/or by reference (URB handle pushed in the payload).

HS Thread Payload (DUAL_PATCH Mode)

GRF DWord	Bits	Description
30:0	Reserved	
R0.6	31	Dereference Thread. This bit is defined to send the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved
	23:0	Thread ID. This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1 KB-aligned offset from the General State Base Address. Format: GeneralStateOffset[31:10]
	9	Reserved
	8:0	FFTID. This ID is assigned by the fixed function unit and is a relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for Implementation Use

GRF DWord	Bits	Description
R0.4	31:5	Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format: SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address. Format: DynamicStateOffset[31:5]
	4	Reserved
	3:0	Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). Programming Note: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:24	Reserved
	23:17	Instance Number. A patch-relative instance number between 0 and InstanceCount - 1. Format = U7
	16:13	Barrier Index. This index is used in any BarrierMsgs sent by this thread to the Gateway. Format = U4
	12	Reserved
	11:0	Reserved
R0.1	31:16	Reserved
	15:0	Patch 1 Patch Data Record URB Return Handle. Format: U14 64B-aligned URB Offset
R0.0	31:16	Reserved
	15:0	Patch 0 Patch Data Record URB Return Handle. Format: U14 64B-aligned URB Offset
The following register is included only if Include PrimitiveID is enabled.		
R1.7:R1.5	31:0	Reserved
R1.4	31:0	Primitive ID 1. This field contains the Primitive ID associated with all instances of input Patch 1 (if present). Format: U32
R1.3:R1.1	31:0	Reserved
R1.0	31:0	Primitive ID 1. This field contains the Primitive ID associated with all instances of input Patch 0. Format: U32
The following register is only included for dispatches that have Include Vertex Handles enabled.		

GRF DWord	Bits	Description
31:16	31:16	Reserved
	15:0	Patch 1 ICP 3 Handle Format: U14 64B-aligned URB Offset
R2.6	31:16	Reserved
	15:0	Patch 1 ICP 2 Handle
R2.5	31:16	Reserved
	15:0	Patch 1 ICP 1 Handle
R2.4	31:16	Reserved
	15:0	Patch 1 ICP 0 Handle
R2.3	31:16	Reserved
	15:0	Patch 0 ICP 3 Handle
R2.2	31:16	Reserved
	15:0	Patch 0 ICP 2 Handle
R2.1	31:16	Reserved
	15:0	Patch 0 ICP 1 Handle
R2.0	31:16	Reserved
	15:0	Patch 0 ICP 0 Handle
[Varies] optional	255:0	Constant Data (optional): Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
[Varies] optional.	255:0	ICP Vertex Data (optional). There can be up to 4 ICPs (vertices) supplied per patch, each with a size defined by the Vertex URB Entry Read Length state. The pushed vertex data is split into upper and lower halves with Patch 0 input vertices in the lower half, and Patch 1 input vertices in the upper half. The pushed data for Patch 0,1 Vertex 0 immediately follows any pushed constant data. The pushed data for Patch 0,1 Vertex 1 immediately follows Vertex 0, and so on.

8_PATCH Payload

The following table shows the layout of the payload delivered to HS threads. Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

Patch object vertex (ICP) data can be passed by value (data pushed in the payload) and/or by reference (URB handle pushed in the payload).

8_PATCH HS Thread Payload

GRF DWord	Bits	Description
R0.7	31	Reserved
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer.** Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address. Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the fixed function unit and is a relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for Implementation Use
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	**Reserved**
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Programming Notes:This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:24	**Reserved**

GRF DWord	Bits	Description
	23:17	**Instance Number.** A patch-relative instance number between 0 and InstanceCount-1. Format = U4
	16:13	**Barrier Index.** This index is to be used in any BarrierMsgs sent by this thread to the Gateway. Format = U4
	12:0	Reserved
R0.1-R0.0	31:0	Reserved
R1.7	31:0	**URB Return Handle for Patch 7** (See R1.0)
R1.6	31:0	**URB Return Handle for Patch 6** (See R1.0)
R1.5	31:0	**URB Return Handle for Patch 5** (See R1.0)
R1.4	31:0	**URB Return Handle for Patch 4** (See R1.0)
R1.3	31:0	**URB Return Handle for Patch 3** (See R1.0)
R1.2	31:0	**URB Return Handle for Patch 2** (See R1.0)
R1.1	31:0	**URB Return Handle for Patch 1** (See R1.0)
R1.0	31:16	Reserved
	15:0	**URB Return Handle 0:** This is the offset of the Patch 0's URB entry, where shading results are to be written. Format: U16 64B-aligned URB Offset
The following register is included only if Include PrimitiveID is enabled.		
R2.7	31:0	**Primitive ID 7.** This field contains the Primitive ID associated with Patch 7 Format: U32
R2.6	31:0	**Primitive ID 6.** This field contains the Primitive ID associated with Patch 6 Format: U32
R2.5	31:0	**Primitive ID 5.** This field contains the Primitive ID associated with Patch 5 Format: U32
R2.4	31:0	**Primitive ID 4.** This field contains the Primitive ID associated with Patch 4 Format: U32
R2.3	31:0	**Primitive ID 3.** This field contains the Primitive ID associated with Patch 3 Format: U32

GRF DWord	Bits	Description
R2.2	31:0	**Primitive ID 2.** This field contains the Primitive ID associated with Patch 2 Format: U32
R2.1	31:0	**Primitive ID 1.** This field contains the Primitive ID associated with Patch 1 Format: U32
R2.0	31:0	**Primitive ID 0.** This field contains the Primitive ID associated with Patch 0 Format: U32
The following registers are included only if Include Vertex Handles is enabled		
Rn.7	31:16	Reserved
	15:0	**Patch 7 ICP 0 Handle**
Rn.6	31:16	Reserved
	15:0	**Patch 6 ICP 0 Handle**
Rn.5	31:16	Reserved
	15:0	**Patch 5 ICP 0 Handle**
Rn.4	31:16	Reserved
	15:0	**Patch 4 ICP 0 Handle**
Rn.3	31:16	Reserved
	15:0	**Patch 3 ICP 0 Handle**
Rn.2	31:16	Reserved
	15:0	**Patch 2 ICP 0 Handle**
Rn.1	31:16	Reserved
	15:0	**Patch 1 ICP 0 Handle**
Rn.0	31:16	Reserved
	15:0	**Patch 0 ICP 0 Handle**
[Rn+1]	255:0	**ICP 1 Handle for Patches 0-7**
[Rn+2]	255:0	**ICP 2 Handle for Patches 0-7**
...		...
[Rn+31]	255:0	ICP 31 Handle for Patches 0-7
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
Varies		**Pushed Vertex Data (optional)** Input data for the 8 patches is located here. Patch 0 (starting with Vertex 0 of Patch 0) data is

GRF DWord	Bits	Description
		passed in DW0 of these GRFs, and Patch 7 data is passed in DW7. The first GRF contains Vertex 0 Element 0 Component 0 for all 8 patches, followed by components 1-3 in the three subsequent GRFs. This is followed by GRFs containing Vertex 0 Element 1 (if it exists), and so on, up to the number of Vertex 0 elements specified by Vertex URB Read Length. This is followed by the data for Vertex 1 for all patches (if it exists), and so on until all relevant vertices are passed. **Programming Note** **Context:** 8_PATCH Payload - Pushed Vertex Data The amount of data passed is limited by the number of GRFs supported by EUs. Software is responsible for comprehending this limit and resorting to the pull model as required.
Rv.7	31:0	**Patch 7 Vertex 0 Element 0 Component 0**
Rv.6	31:0	**Patch 6 Vertex 0 Element 0 Component 0**
Rv.5	31:0	**Patch 5 Vertex 0 Element 0 Component 0**
Rv.4	31:0	**Patch 4 Vertex 0 Element 0 Component 0**
Rv.3	31:0	**Patch 3 Vertex 0 Element 0 Component 0**
Rv.2	31:0	**Patch 2 Vertex 0 Element 0 Component 0**
Rv.1	31:0	**Patch 1 Vertex 0 Element 0 Component 0**
Rv.0	31:0	**Patch 0 Vertex 0 Element 0 Component 0**
Rv+1	31:0	**Patch 0-7 Vertex 0 Element 0 Component 1**
...		and so on...

Tessellation Engine (TE) Stage

When enabled, the Tessellation Engine (TE) stage performs fixed-function domain tessellation (decomposition into smaller objects) of incoming patches, as referenced by an HS-generated input PDR handle and as controlled by TE state and Tessellation Factors (TFs) read from the Patch URB Entry (patch record). The TE stage is entirely fixed-function and does not spawn threads.

The fixed-function tessellation algorithm is considered an implementation detail and is therefore beyond the scope of this document. That detail includes both the order of output topologies as well as the order of vertices (domain points) within the output topologies. Only a high-level overview is provided to describe how the (few) state variables can be used to control aspects of tessellation behavior. The implementation will generate deterministic results (given the same exact inputs it will produce exactly the same outputs).

Several domain types (QUAD, TRI, and ISOLINE) are supported. Depending on the domain type, the TE stage outputs the required point/line/triangle topologies including a domain point per vertex. These topologies will be output to the DS stage, where the domain points will be converted to 3D object vertices, resulting in 3D objects as typically input to the 3D pipeline when HOS tessellation is not used.

When tessellation is disabled, all topologies (including patchlist topologies) simply pass through to the GS stage. When tessellation is enabled, only patchlist topologies should be issued to the pipeline, else behavior is UNDEFINED. The MI_TOPOLOGY_FILTER command can be used to ensure this happens, i.e., it can be used to have the Command Stream ignore 3DPRIMITIVE commands that do not match a specific topology type.

Enabling tessellation is accomplished by enabling the HS/TE/DS stages in specific combinations. Those valid combinations are described in the table below.

Valid Tesellation Enabled Configurations
To enable tessellation, the HS, TE, and DS stages must be enabled and disabled together. Other configurations will result in behavior that is UNDEFINED.

State

This section contains the state registers for the Tessellation Engine.

3DSTATE_TE

Functions

Patch Culling

Normally, if any “outside” TF is <= 0.0 or NaN, the entire patch is culled at the TE stage.

Inside TFs are not used to cull patches.

Tessellation Factor Limits

After the Patch Culling test is performed, the TessFactors undergo a min() clamp to either the **MaxTessFactorOdd** (for FRACTIONAL_ODD partitioning) or **MaxTessFactorNotOdd** (for FRACTIONAL_EVEN or INTEGER partitioning). Exception: If the ISOLINE domain is specified, the LineDensity TessFactor will be clamped to the **MaxFactorNotOdd** value even if FRACTIONAL_ODD partitioning is specified).

Partitioning

The Partitioning state controls how the TFs are used to divide their corresponding edges.

Partitioning Mode	Definition
INTEGER	The edge is divided into an integral number of equal segments (given some fixed-point tolerance). After clamping, the TF is rounded up to an integer value. The edge is divided into that many equal segments.

Partitioning Mode	Definition
EVEN_FRACTIONAL	The edge is divided into an *even* number of possibly-unequal segments. The total number of segments is determined by rounding up the post-clamped TF to an even number. More specifically, the edge is divided exactly in half. Like the endpoints of the edge, the midpoint of the edge is by definition a tessellation point. Each half contains some number of equal segments and possibly one smaller segment. The size of the smaller segment is determined by the position of the TF value within the range defined by the TF rounded down and up to even numbers. The closer the TF is to the smaller value, the smaller the segment size is. When the TF reaches the smaller even value, the smaller segment disappears. The closer the TF gets to the larger even value, the closer the smaller segment size approaches the size of the other segments. When the TF reaches the larger even value, all segments are equal. The position of the smaller segment along the half edge varies as a function of the TF value.
ODD_FRACTIONAL	The edge is divided into an *odd* number of possibly-unequal segments. The tessellation scheme is very similar to EVEN_FRACTIONAL partitioning, except that the edge midpoint is not included as a tessellation point. This, and the fact that the tessellation points are mirrored about the edge midpoint, causes an "odd" segment (which may or may not be the "smaller" segment) to straddle the edge midpoint, therefore resulting in the number of segments for the edge always being odd.

Domain Types and Output Topologies

The major (if only) task of the TE stage is to tessellate a 2D (u,v) domain region, as selected by the Domain state, into some number of 2D object topologies. (If the patch is culled, that number may be zero). The options for Domain state are:

- QUAD: A square 2D region within a u,v Cartesian (rectanguar) space. The region extends from the origin to u=1 and v=1. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- TRI: A triangular 2D region with u,v,w barycentric (areal) coordinates. The three edges correspond to u=0, v=0, and w=0 boundaries. In barycentric coordinates, w = 1 – u – v, therefore points within the region are fully defined as 2D (u,v) coordinates. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- ISOLINE: A series of points within a QUAD domain, where the points lie on lines parallel to the u axis and extending from [0,1) in the v direction. Either the segmented lines (linestrips) or individual point topologies can be output.

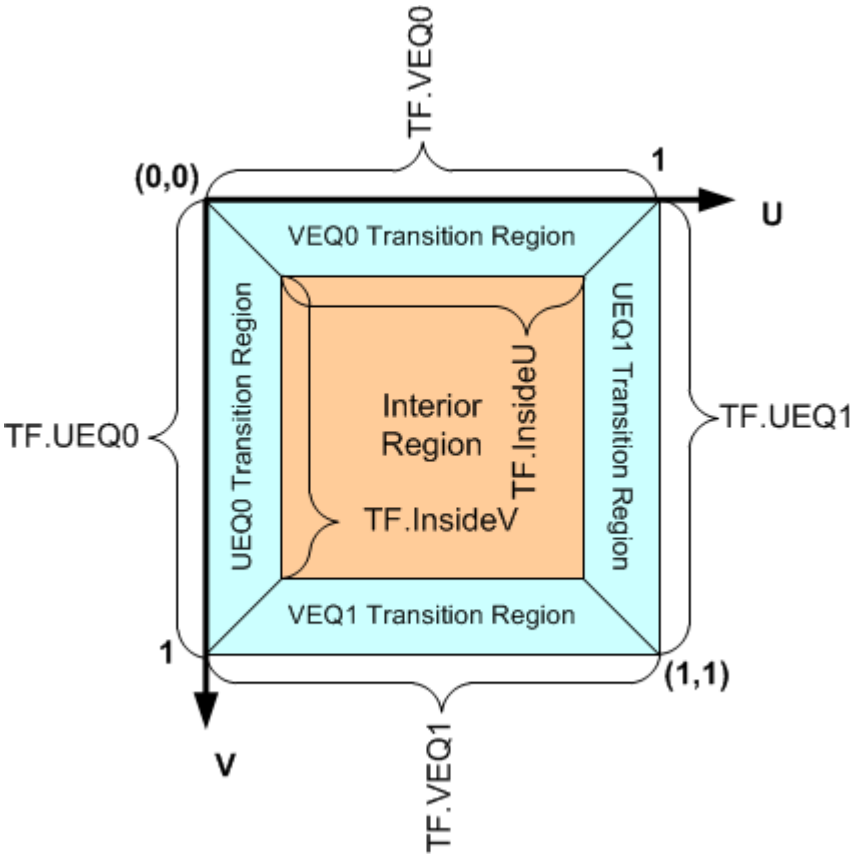
QUAD Domain Tessellation

The four "outside" TFs (TF.UEQ0, TF.VEQ0, TF.UEQ1, TF.VEQ1) are used to specify the level of tessellation along the four corresponding edges of the 2D quad domain. The two "inside" TFs (TF.InsideU, TF.InsideV)

are used to determine the level of tessellation within a 2D “interior” region. Typically the interior region appears as a “regularly-tessellated 2D grid”, however under certain conditions the interior region may collapse in which case only the outside TFs are relevant.

In general, a transition region exists between each edge of the interior region and the corresponding outside edge. The topologies generated for these regions effectively “stitch together” locations along the outside and inside edges, as each of these edges can contain a different number of tessellated segments. In the case where all TFs in a given direction (e.g., TF.VEQ0, TF.InsideU, and TF.VEQ1) are the same value, it appears as if the regularly-tessellated interior region extends all the way to the outside edges. If this condition simultaneously exists for both u and v directions, the entire domain will appear to be tessellated into a regular grid, with no noticeable transition regions.

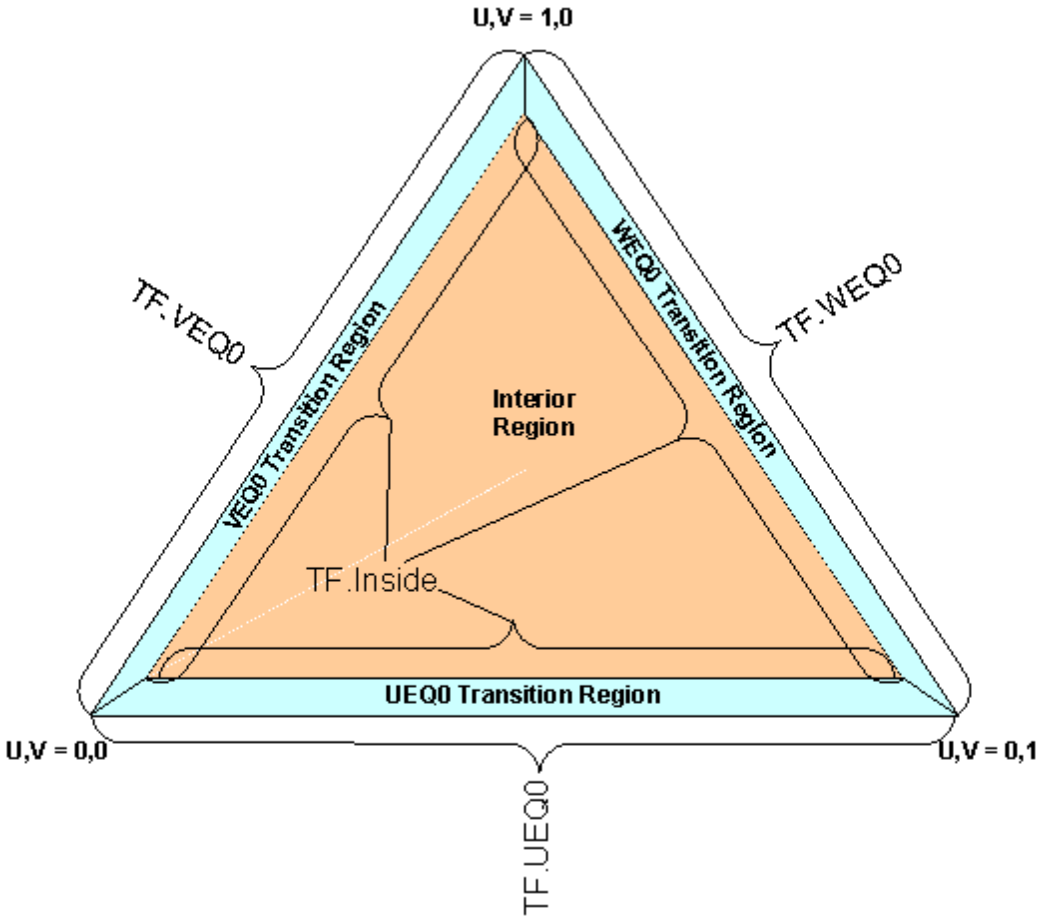
QUAD Domain



TRI Domain Tessellation

Tessellation of the TRI domain is similar to the QUAD domain, except only three outside edges/TFs are used, and the tessellation of the interior region is controlled by a single TF.

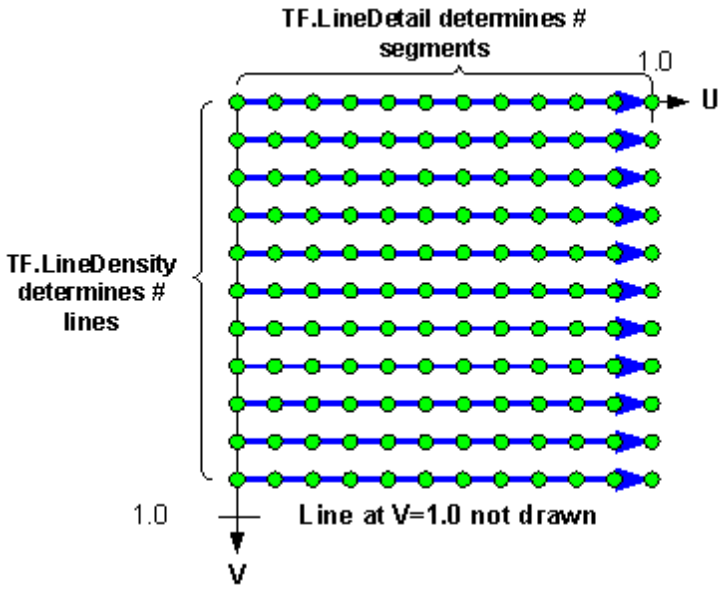
TRI Domain



ISOLINE Domain Tessellation

Tessellation of the ISOLINE domain is different but much simpler than QUAD and TRI domains. The TF.LineDetail TF controls how finely the U direction is tessellated, while the TF.LineDensity TF controls how finely the V direction is tessellated. When LINE output topology is selected, a series of segmented lines parallel to the U axis (constant V) are output. When POINT output topology is selected, only the line segment endpoints are output (as point objects). In either case there is no topology output for the V=1 edge, which avoids overlapping lines for adjacent patches.

ISOLINE Domain



Domain Shader (DS) Stage

The DS stage is very similar to the VS stage in that it is responsible for dispatching EU threads to shade vertices and maintaining a cache (with reference counts) of the shaded vertex outputs of these threads. Major differences are as follows:

- The DS receives topologies with "domain points" instead of vertices. The only data specific to a domain point are its U,V coordinates. These coordinates (plus a default or computed W coordinate) are passed directly in the DS thread payload. There is no other vertex-specific "input vertex data".
- The concatenation of the domain point U,V coordinates (vs. a vertex index) is used as the cache tag.
- The cache is invalidated between patches.

The DS stage accepts state information via the inline 3DSTATE_DS command.

State

This section contains the state registers for the Domain Shader.

3DSTATE_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT(Body)

3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_URB_DS

Functions

This topic is currently under development.

SIMD4x2 Thread Execution

Description
A DS kernel assumes it is to operate on up to eight domain points in parallel using the EU's SIMD8 execution model, or on two domain points in parallel (using the EU's SIMD4x2 execution model).

Refer to the ISA chapters for specifics on writing kernels that operate in SIMD4x2 fashion.

DS threads must always write the destination URB handles passed in the payload. DS threads are not permitted to request additional destination handles. Refer to 3D Pipeline Stage Overview (*3D Overview*) for details on how destination vertices are written and any required contents/formats.

DS threads must signal thread termination on the last message output to the URB shared function.

DUAL_PATCH Thread Execution

When Dispatch Mode is set to SIMD8_SINGLE_OR_DUAL_PATCH mode, both the KSP and DUAL_PATCH KSP kernels are enabled. The DS stage decides whether to spawn a SINGLE_PATCH (KSP) or DUAL_PATCH thread dynamically, based on the number of domain points associated with patches. (See Implementation Note below).

- The KSP kernel operates exactly like when SIMD8_SINGLE_PATCH mode is set. Up to 8 domain points for a single patch are processed by the DS thread, which operates in SIMD8 fashion.
- The DUAL KSP kernel uses a hybrid SIMD8 execution mode. The 8 execution channels are divided into 4 upper channels associated with Patch 1, and 4 lower channels associated with Patch 0. Patch data is passed in SIMD4x2 layout, with Patch 1 data (Primitive ID, pushed URB data) in the upper 4 channels, and Patch 0 data in the lower 4 channels. The kernel operates much like SIMD8_SINGLE_PATCH mode, though it needs to access the appropriate SIMD4 patch data.

Implementation Note: Kernel selection is as follows: If a patch requires more than 4 domain points to be shaded, SIMD8_SINGLE_PATCH threads are spawned until 4 or fewer domain points remain. These domain points (if any exist) are held pending until the next patch is received. Likewise, if a patch requires 4 or fewer total domain points, those domain points are held pending. In either case, if the subsequent patch requires 4 or fewer domain points to be shaded, a SIMD8_DUAL_PATCH thread is spawned to shade both sets of 4 or fewer domain points. If the subsequent patch requires more than 4 domain points, the (4 or fewer) buffered domain points of the previous patch are shaded via a SIMD8_SINGLE_PATCH thread, and the cycle continues.

Statistics Gathering

The DS stage maintains the DS_INVOCATIONS statistics counter, which counts the number of incoming domain points, irrespective of cache hit/miss. Note that this is different than VS_INVOCATIONS, which counts shader invocations and therefore doesn't count cache hits.

Payloads

This topic is currently under development.

SIMD4x2 Payload

The following table describes the payload delivered to DS threads.

DS Thread Payload (SIMD4x2)

DWord	Bits	Description
R0.7	31	
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use. Format = U9
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>R0.3</td><td>31:5</td><td>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.
Format = DynamicStateOffset[31:5]</td></tr>
<tr><td></td><td>4</td><td>Reserved</td></tr>
<tr><td></td><td>3:0</td><td>Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).
Format = U4 power of two (in excess of 10)
Range = [0,11] indicating [1K Bytes, 2M Bytes]</td></tr>
<tr><td>R0.2</td><td>31:0</td><td>Reserved: delivered as zeros (reserved for message header fields)</td></tr>
<tr><td>R0.1</td><td>31:26</td><td>Reserved</td></tr>
<tr><td></td><td>25:16</td><td><table><tr><th>Description</th></tr><tr><td>Reserved</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td>Reserved</td></tr>
<tr><td></td><td>13:0</td><td>URB Return Handle 1: This is the URB handle where Vertex 1 data (the EU’s upper channels (DWords 7:4)) results are to be stored.
If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread’s Channel Mask).
Format:<table><tr><th>Format</th></tr><tr><td>U14 handle (512-bit granular)</td></tr></table></td></tr>
<tr><td>R0.0</td><td>31:26</td><td>Reserved</td></tr>
<tr><td></td><td>25:16</td><td><table><tr><th>Description</th></tr><tr><td>Reserved</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td>Reserved</td></tr>
<tr><td></td><td>13:0</td><td>URB Return Handle 0: This is the URB handle where Vertex 0 data (the EU’s lower channels (DWords 3:0)) results are to be stored.
Format:<table><tr><th>Format</th></tr><tr><td>U14 handle (512-bit granular)</td></tr></table></td></tr>
</table>

DWord	Bits	Description
R1.7	31:0	**PrimitiveID.** This is the 32-bit PrimitiveID value associated with the patch. It is common to all output vertices resulting from the tessellation of the patch. Format: U32
R1.6	31:0	**Domain Point 1 W Coordinate.** (See Domain Point 0 W Coordinate) Format: FLOAT32
R1.5	31:0	**Domain Point 1 V Coordinate.** (See Domain Point 0 V Coordinate) Format: FLOAT32
R1.4	31:0	**Domain Point 1 U Coordinate.** (See Domain Point 0 U Coordinate) Format: FLOAT32
R1.3	31:14	Reserved
	13:0	**Patch URB Handle.** This is the URB handle of the Patch Record (common to both vertices). Format: Format U14 handle)
R1.2	31:0	**Domain Point 0 W Coordinate.** If **Compute W Coordinate Enable** is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R1.1	31:0	**Domain Point 0 V Coordinate.** V coordinate associated with Domain Point 0. Format: FLOAT32
R1.0	31:0	**Domain Point 0 U Coordinate.** U coordinate associated with Domain Point 0. Format: FLOAT32
Varies [Optional]	255:0	**Constant Data (optional).** Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_DS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.

DWord	Bits	Description
Varies [Optional]	255:0	**Patch URB Data (optional).** Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS). The Patch Data arrives in a non-interleaved format.

SIMD8 Payload

The following table describes the payload delivered to DS threads.

DS Thread Payload (SIMD8)

DWord	Bits	Description
R0.7	31	Reserved
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	Reserved

DWord	Bits	Description
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:0	**PrimitiveID.** This is the 32-bit PrimitiveID value associated with the patch. It is common to all output domain points resulting from the tessellation of the patch. Format: U32
R0.0	31:27	Reserved
	26:16	**Description** Reserved
	15:0	**Patch URB Offset.** This is the offset within the URB where the patch data is stored. Format: U14 64B-granular offset into the URB
R1.7	31:0	**Domain Point 7 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.6	31:0	**Domain Point 6 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.5	31:0	**Domain Point 5 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.4	31:0	**Domain Point 4 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.3	31:0	**Domain Point 3 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.2	31:0	**Domain Point 2 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.1	31:0	**Domain Point 1 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.0	31:0	**Domain Point 0 U Coordinate.** U coordinate associated with Domain Point 0. Format: FLOAT32
R2.7	31:0	**Domain Point 7 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.6	31:0	**Domain Point 6 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.5	31:0	**Domain Point 5 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.4	31:0	**Domain Point 4 V Coordinate.** (See Domain Point 0 V Coordinate.)

DWord	Bits	Description
R2.3	31:0	**Domain Point 3 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.2	31:0	**Domain Point 2 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.1	31:0	**Domain Point 1 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.0	31:0	**Domain Point 0 V Coordinate.** V coordinate associated with Domain Point 0. Format: FLOAT32
R3.7	31:0	**Domain Point 7 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.6	31:0	**Domain Point 6 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.5	31:0	**Domain Point 5 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.4	31:0	**Domain Point 4 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.3	31:0	**Domain Point 3 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.2	31:0	**Domain Point 2 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.1	31:0	**Domain Point 1 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.0	31:0	**Domain Point 0 W Coordinate.** If Compute W Coordinate Enable is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R4.7	31:0	**Domain Point 7 URB Return Handle.** (See R4.0.)
R4.6	31:0	**Domain Point 6 URB Return Handle.** (See R4.0.)
R4.5	31:0	**Domain Point 5 URB Return Handle.** (See R4.0.)
R4.4	31:0	**Domain Point 4 URB Return Handle.** (See R4.0.)
R4.3	31:0	**Domain Point 3 URB Return Handle.** (See R4.0.)
R4.2	31:0	**Domain Point 2 URB Return Handle.** (See R4.0.)
R4.1	31:0	**Domain Point 1 URB Return Handle.** (See R4.0.)
R4.0	31:16	Reserved
	15:0	**Domain Point 0 URB Return Handle.** This is the offset within the URB where domain point 0 is to be stored. Format: U14 64B-granular offset into the URB
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
Varies [Optional]	255:0	**Patch URB Data (optional).** Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS).

DUAL_PATCH Payload

The following table describes the payload delivered to DS threads.

DUAL_PATCH DS Thread Payload (SIMD8)

DWord	Bits	Description
R0.7	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]

DWord	Bits	Description
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:27	Reserved
	26:16	Description Reserved
	15:0	**Patch 1 URB Offset.** This is the offset within the URB where the Patch 1 data is stored. Format: U14 64B-granular offset into the URB
R0.0	31:27	Reserved
	26:16	Description Reserved
	15:0	**Patch 0 URB Offset.** This is the offset within the URB where the Patch 0 data is stored. Format: U14 64B-granular offset into the URB
R1.5-7	31:0	Reserved
R1.4	31:0	**Patch 1 PrimitiveID.** This is the 32-bit PrimitiveID value associated with Patch 1. Format: U32
R1.1-3	31:0	Reserved
R1.0	31:0	**Patch 0 PrimitiveID.** This is the 32-bit PrimitiveID value associated with Patch 0. Format: U32
R2.7	31:0	**Patch 1 Domain Point 3 U Coordinate.** (See R2.0.)
R2.6	31:0	**Patch 1 Domain Point 2 U Coordinate.** (See R2.0.)
R2.5	31:0	**Patch 1 Domain Point 1 U Coordinate.** (See R2.0.)
R2.4	31:0	**Patch 1 Domain Point 0 U Coordinate.** (See R2.0.)
R2.3	31:0	**Patch 0 Domain Point 3 U Coordinate.** (See R2.0.)
R2.2	31:0	**Patch 0 Domain Point 2 U Coordinate.** (See R2.0.)
R2.1	31:0	**Patch 0 Domain Point 1 U Coordinate.** (See R2.0.)
R2.0	31:0	**Patch 0 Domain Point 0 U Coordinate.** U coordinate associated with Domain Point 0 of Patch 0. Format: FLOAT32
R3.7	31:0	**Patch 1 Domain Point 3 V Coordinate.** (See R3.0.)
R3.6	31:0	**Patch 1 Domain Point 2 V Coordinate.** (See R3.0.)
R3.5	31:0	**Patch 1 Domain Point 1 V Coordinate.** (See R3.0.)
R3.4	31:0	**Patch 1 Domain Point 0 V Coordinate.** (See R3.0.)

DWord	Bits	Description
R3.3	31:0	**Patch 0 Domain Point 3 V Coordinate.** (See R3.0.)
R3.2	31:0	**Patch 0 Domain Point 2 V Coordinate.** (See R3.0.)
R3.1	31:0	**Patch 0 Domain Point 1 V Coordinate.** (See R3.0.)
R3.0	31:0	**Patch 0 Domain Point 0 V Coordinate.** V coordinate associated with Domain Point 0. Format: FLOAT32
R4.7	31:0	**Patch 1 Domain Point 3 W Coordinate.** (See R4.0.)
R4.6	31:0	**Patch 1 Domain Point 2 W Coordinate.** (See R4.0.)
R4.5	31:0	**Patch 1 Domain Point 1 W Coordinate.** (See R4.0.)
R4.4	31:0	**Patch 1 Domain Point 0 W Coordinate.** (See R4.0.)
R4.3	31:0	**Patch 0 Domain Point 3 W Coordinate.** (See R4.0.)
R4.2	31:0	**Patch 0 Domain Point 2 W Coordinate.** (See R4.0.)
R4.1	31:0	**Patch 0 Domain Point 1 W Coordinate.** (See R4.0.)
R4.0	31:0	**Patch 0 Domain Point 0 W Coordinate.** If Compute W Coordinate Enable is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R5.7	31:0	**Patch 1 Domain Point 3 URB Return Handle.** (See R5.0.)
R5.6	31:0	**Patch 1 Domain Point 2 URB Return Handle.** (See R5.0.)
R5.5	31:0	**Patch 1 Domain Point 1 URB Return Handle.** (See R5.0.)
R5.4	31:0	**Patch 1 Domain Point 0 URB Return Handle.** (See R5.0.)
R5.3	31:0	**Patch 0 Domain Point 3 URB Return Handle.** (See R5.0.)
R5.2	31:0	**Patch 0 Domain Point 2 URB Return Handle.** (See R5.0.)
R5.1	31:0	**Patch 0 Domain Point 1 URB Return Handle.** (See R5.0.)
R5.0	31:16	Reserved
	15:0	**Patch 0 Domain Point 0 URB Return Handle.** This is the offset within the URB where Patch 0 Domain Point 0 is to be stored. Format: U14 64B-granular offset into the URB
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.

DWord	Bits	Description
		Patch 0,1 URB Data follows (optional). This data is read from the URB and pushed in the payload. The amount of data provided for each patch (which may be 0) is defined by the Patch URB Entry Read Length state (3DSTATE_DS). The data is read from the URB starting at the Patch URB Entry Read Offset into each patch, so leading data with the Patch URB entries can be skipped over. Patch 1 data is passed in the upper 128 bits, while Patch 0 data is passed in the lower 128 bits. This is similar to how URB data is pushed into SIMD4x2 kernels (VS, GS, etc.).
[Varies] optional	255:128	**Patch 1 URB Data.**
	127:0	**Patch 0 URB Data.**

Geometry Shader (GS) Stage

GS Stage Overview

The GS stage of the 3D Pipeline converts objects within incoming primitives into new primitives through use of a spawned thread. When enabled, the GS unit buffers incoming vertices, assembles the vertices of each individual object within the primitives, and passes those object vertices (along with other data) to the graphics subsystem for processing by a GS thread.

When the GS stage is disabled, vertices flow through the unit unmodified.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Pipeline* chapter for a general description of a 3D Pipeline stage, as much of the GS stage operation and control falls under these "common" functions. I.e., most stage state variables and GS thread payload parameters are described in *3D Pipeline*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the GS stage, and any exceptions the GS stage exhibits with respect to common FF unit functions.

State

This sections contains the state registers for the Geometry Shader.

The state used by GS is defined with this inline state packet.

3DSTATE_GS

3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT(Body)

3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_URB_GS

Functions

Object Staging

The GS unit's Object Staging Buffer (OSB) accepts primitive topologies as a stream of incoming vertices, and spawns a thread for each individual object within the topology.

Thread Request Generation

Object Vertex Ordering

The following table defines the number and order of object vertices passed in the Vertex Data portion of the GS thread payload, assuming an input topology with *N* vertices. The ObjectType passed to the thread is, by default, the incoming PrimTopologyType. Exceptions to this rule (for the TRISTRIP variants) are called out.

The following table also shows which vertex is selected to provide PrimitiveID (bold, underlined vertex number). In general, the vertex selected is the last vertex for non-adjacent prims, and the next-to-last vertex for adjacent prims. Note, however, that there are exceptions:

- reorder-enabled TRISTRIP[_REV], TRISTRIP_ADJ
- "odd-numbered" objects in TRISTRIP_ADJ

PrimTopologyType	Order of Vertices in Payload	
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
POINTLIST	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
POINTLIST_BF	N/A	
LINELIST (N is multiple of 2)	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
LINELIST_ADJ (N is multiple of 4)	[0] = (0,1,2,3); [1] = (4,5,6,7); ...; [(N/4)-1)] = (N-4,N-3,N-2,N-1)	
LINESTRIP (N >= 2)	[0] = (0,1); [1] = (1,2); ...; [N-2] = (N-2,N-1)	
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT (N >= 4)	[0] = (0,1,2,3); [1] = (1,2,3,4); ...; [N-4] = (N-4,N-3,N-2,N-1)	LINESTRIP_ADJ_CONT is added for BDW+. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINESTRIP_BF	N/A	
LINESTRIP_CONT	Same as LINESTRIP	Handled same as LINESTRIP
LINESTRIP_CONT_BF	Same as LINESTRIP	Handled same as LINESTRIP
LINELOOP (N >= 2)	[0] = (0,1); [1] = (1,2); [N] = (N-1,0);	Not supported after GS.
TRILIST (N is multiple of 3)	[0] = (0,1,2); [1] = (3,4,5); ...; [(N/3)-1] = (N-3,N-2,N-1)	
RECTLIST, RECTLIST_SUBPIXEL	Same as TRILIST	Handled same as TRILIST
TRILIST_ADJ (N is multiple of 6)	[0] = (0,1,2,3,4,5); [1] = (6,7,8,9,10,11); ...; [(N/6)-1] = (N-6,N-5,N-4,N-3,N-2,N-1)	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
TRISTRIP (Reorder Leading) (N >= 3)	[0] = (0,1,2); {TRISTRIP} [1] = (1,3,2); {TRISTRIP_REV} [k even] = (k,k+1,k+2) {TRISTRIP} [k odd] = (k,k+2,k+1) {TRISTRIP_REV} [N-3] = (see above)	"Odd" triangles have vertices reordered and identified as TRISTRIP to inform the thread.
TRISTRIP (Reorder Trailing) (N >= 3)	[0] = (0,1,2) {TRISTRIP} [1] = (2,1,3) {TRISTRIP_REV}; ... [k even] = (k,k+1,k+2) {TRISTRIP} [k odd] = (k+1,k,k+2) {TRISTRIP_REV} [N-3] = (see above)	"Odd" triangles have vertices reordered and identified as TRISTRIP_REV to inform the thread.
TRISTRIP_REV (Reorder Leading) (N >= 3)	[0] = (0,2,1) {TRISTRIP_REV}; [1] = (1,2,3) {TRISTRIP}; ...; [k even] = (k,k+2,k+1) {TRISTRIP_REV} [k odd] = (k,k+1,k+2) {TRISTRIP} [N-3] = (see above)	"Even" triangles have vertices reordered and identified as TRISTRIP to inform the thread.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
TRISTRIP_REV (Reorder Trailing) (N >= 3)	[0] = (1,0,2) {TRISTRIP_REV} [1] = (1,2,3) {TRISTRIP}; ...; [k even] = (k+1,k,k+2,) {TRISTRIP_REV} [k odd] = (k,k+1,k+2) {TRISTRIP} [N-3] = (see above)	"Even" triangles have vertices reordered and identified as TRISTRIP_REV to inform the thread.
TRISTRIP_ADJ (Reorder Leading) (N >= 6)	N = 6 or 7: [0] = (0,1,2,5,4,3) N = 8 or 9: [0] = (0,1,2,6,4,3); [1] = (2,5,6,7,4,0); ...; N >= 10: [0] = (0,1,2,6,4,3); [1] = (2,5,6,8,4,0); ...; [k>1, even] = (2k,2k-2, 2k+2, 2k+6,2k+4, 2k+3); [k>2, odd] = (2k, 2k+3, 2k+4, 2k+6, 2k+2, 2k-2);...; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1,N-2,N-3); [(N/2)-3, odd] = (N-6,N-3,N-2,N-1,N-4,N-8);	Objects have vertices reordered.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
TRISTRIP_ADJ (Reorder Trailing) (N >= 6)	N = 6 or 7: [0] = (0,1,2,5,4,3) N = 8 or 9: [0] = (0,1,2,6,4,3); [1] = (4,0,2,5,6,7); ...; N >= 10: [0] = (0,1,2,6,4,3); [1] = (4,0,2,5,6,8); ...; [k>1, even] = (2k,2k-2, 2k+2, 2k+6,2k+4, 2k+3); [k>2, odd] = (2k+2, 2k-2, 2k, 2k+3, 2k+4, 2k+6);...; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1,N-2,N-3); [(N/2)-3, odd] = (N-4,N-8,N-6,N-3,N-2,N-1);	OpenGL ordering rules (last non-adjacent vertex is the last – aka provoking – vertex of the triangle). Even triangles have the same ordering as Leading Vertex, odd triangle ordering is different (rotated 2 vertices).
TRIFAN (N > 2)	[0] = (0,1,2); [1] = (0,2,3); ...; [N-3] = (0, N-2, N-1);	
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON, POLYGON_CONT	Same as TRIFAN	POLYGON_CONT is added for BDW+. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
QUADLIST	[0] = (0,1,2,3); [1] = (4,5,6,7); ...; [(N/4)-1] = (N-4,N-3,N-2,N-1);	Not supported after GS. : QUADLIST primitives are converted into POLYGONS in VF, and therefore never reach the GS.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
QUADSTRIP	[0] = (0,1,3,2); [1] = (2,3,5,4); ... ; [(N/2)-2] = (N-4,N-3,N-1,N-2);	Not supported after GS. : QUADSTRIP primitives are converted into POLYGONS in VF, and therefore never reach the GS.
: PATCHLIST_1 PATCHLIST_2 PATCHLIST_3..32	[0] = (0); [1] = (1); ...; [N-2] = (N-2); [0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1) similar to above	

Thread Execution

A GS thread is capable of performing arbitrary algorithms given the thread payload (especially vertex) data and associated data structures (binding tables, sampler state, etc.) as input. Output can take the form of vertices output to the FF pipeline (at the GS unit) and/or data written to memory buffers via the DataPort.

The primary usage models for GS threads include (possible combinations of):

- Compiled application-provided "GS shader" programs, specifying an algorithm to convert the vertices of an input object into some output primitives. For example, a GS shader may convert lines of a line strip into polygons representing a corresponding segment of a blade of grass centered on the line. Or it could use adjacency information to detect silhouette edges of triangles and output polygons extruding out from the those edges. Or it could output absolutely nothing, effectively terminating the pipeline at the GS stage.
- Driver-generated instructions used to write pre-clipped vertices into memory buffers (see Stream Output below). This may be required whether or not an app-provided GS shader is enabled.
- Driver-generated instructions usedto emulate API functions not supported by specialized hardware. These functions might include (but are not limited to):
    - Conversion of API-defined topologies into topologies that can be rendered (e.g., LINELOOP➔LINESTRIP, POLYGON➔TRIFAN, QUADs➔TRIFAN, etc.)

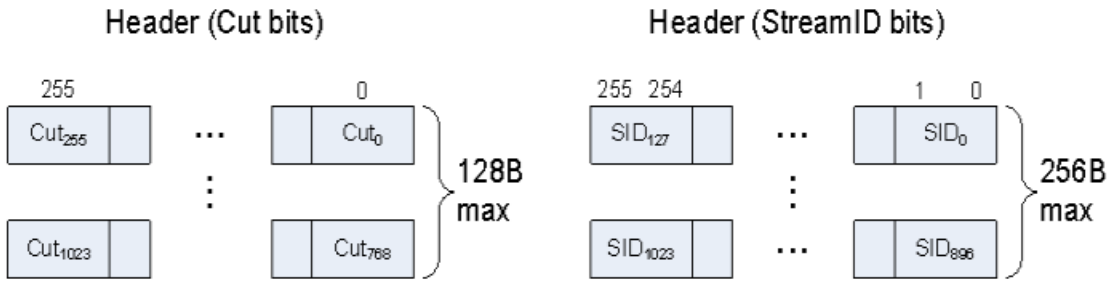
- Emulation of "Polygon Fill Mode", where incoming polygons can be converted to points, lines (wireframe), or solid objects.
- Emulation of wide/sprite points.

When rendering is required, concurrent GS threads must use the FF_SYNC message (URB shared function) to request an initial VUE handle and synchronize output of VUEs to the pipeline (see *URB* in *Shared Functions*). Only one GS thread can be outputting VUEs to the pipeline at a time. To achieve parallelism, GS threads should perform the GS shader algorithm (along with any other required functions) and buffer results (either in the GRF or scratch memory) before issuing the FF_SYNC message. The issuing GS thread is stalled on the FF_SYNC writeback until it is that thread's turn to output VUEs. As only one GS thread at a time can output VUEs, the post-FF_SYNC output portion of the kernel should be optimized as much as possible to maximize parallelism.

Thread Execution

GS URB Entry

All outputs of a GS thread are stored in the single GS thread output URB entry. Cut (1 bit/vertex) or StreamID (2 bits/vertex) bits are packed into an optional 1-8 32B header. The **Control Data Format** and **Control Data Header Size** states specify the size and contents of the header data (if any).



Following the optional header is a variable number of 16B or 32B-aligned/granular vertices:

- When rendering is DISABLED, typically output vertices are 32B-aligned, with the exception of 16B-alignment for vertices <= 16B in length.
  - The absolute worst case size comes from three DW scalars output per vertex. If these are, say, three ".x" outputs, you need to store each DW in a 128b (16B) element, plus another pad 16B to keep the 32B alignment. So you require 4*16B = 64B/vertex. You have to have room for 1024 scalars / 3 scalar/vtx = 341 vertices. 341*64B = 21,824B. Then add 96B to hold 2b/vtx streamID and you get 21,920B entries.
- When rendering is ENABLED, each output vertex is 32B-aligned. Here the vertex header and vertex 'position' are required and therefore the minimum size vertex is 32B.
  - Here the worst case size isn't as bad as render-disabled, as you have to have a 4DW position output, plus any additional output. So, say you output 5 DW per vertex. You need 64B/vertex (16B vtx header, 16B position, 16B for the 2nd element, and 16B of pad). You have to have

room for 1024 scalars / 5 = 204 vertices. 204*64 = 13,056B. Then add 64B to hold 2b/vtx streamID and you get 13,120B entries.

The size of the URB entry should be based on the declared maximum # of output vertices and the declared output vertex size (the union of per-stream vertex structures, if required).

GS URB Entry - Output Vertex Count

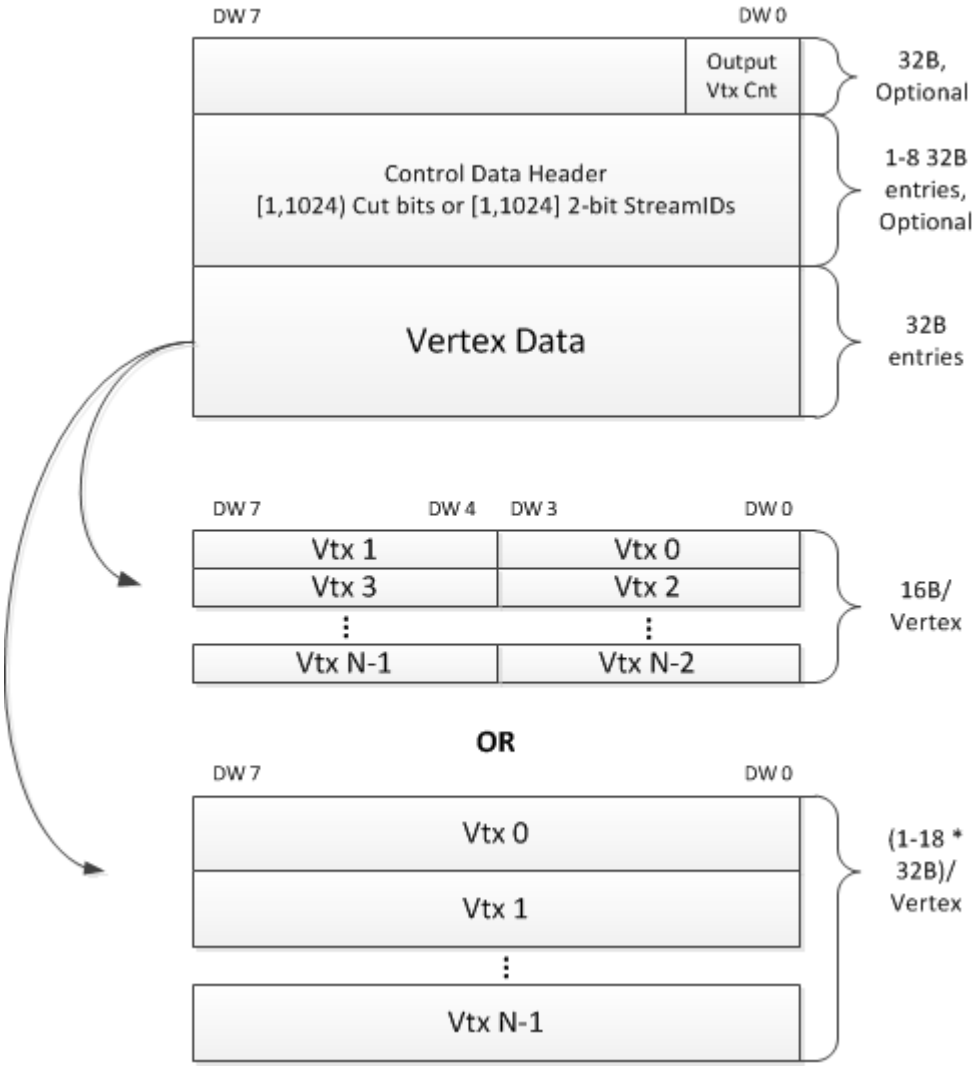
The GS URB entry is the same as in the two previous generations with the following exception: If **Static Output** (3DSTATE_GS) is clear, the URB entry starts with a 32B OUTPUT_VERTEX_COUNT structure as defined below. Ths control header (if present) immediately follows this structure. If **Static Output** is set, the control header (if present) appears at the very start of the URB entry (as described above).

GS OUTPUT_VERTEX_COUNT

DWord	Bit	Description
7:6	31:0	Reserved
0	31:16	Reserved
	15:0	**Output Vertex Count.** Indicates the number of vertices output from this GS shader invocation. Format = U16 Range: [0:1024]

This structure (if present) increases the maximum URB entry sizes (described above) by 32B.

The following diagram illustrates the possible layouts of a GS URB Entry:



GS Output Topologies

The following table lists which primitive topology types are valid for output by a GS thread.

PrimTopologyType	Supported for GS Thread Output?
LINELIST	Yes
LINELIST_ADJ	No
LINESTRIP	Yes
LINESTRIP_ADJ	No
LINESTRIP_BF	Yes
LINESTRIP_CONT	Yes
LINESTRIP_CONT_BF	Yes
LINELOOP	No
POINTLIST	Yes

PrimTopologyType	Supported for GS Thread Output?
POINTLIST_BF	Yes
POLYGON	Yes
QUADLIST	No
QUADSTRIP	No
RECTLIST	Yes
TRIFAN	Yes
TRIFAN_NOSTIPPLE	Yes
TRILIST	Yes
TRILIST_ADJ	No
TRISTRIP	Yes
TRISTRIP_ADJ	No
TRISTRIP_REV	Yes
PATCHLIST_xxx	Yes

GS Output StreamID

When the **GS Enable** is DISABLED, output vertices are assigned a StreamID = 0;

When the **GS Enable** is ENABLED, output vertices are assigned a StreamID = **Default StreamID** under the following conditions:

- **Control Data Format** = 0, or
- **Control Data Format** > 0 and **Control Data Format** = GSCTL_CUT

When the GS is enabled, **Control Data Format** > 0 and **Control Data Format** = GSCTL_SID, output vertices are assigned a StreamID as programmed in the Control Data output by the thread.

Primitive Output

(This section refers to output from the GS unit to the pipeline, not output from the GS thread)

The GS unit will output primitives (either passed-through or generated by a GS thread) in the proper order. This includes the buffering of a concurrent GS thread's output until the preceding GS thread terminates. Note that the requirement to buffer subsequent GS thread output until the preceding GS thread terminates has ramifications on determining the number of VUEs allocated to the GS unit and the number of concurrent GS threads allowed.

Statistics Gathering

There are a number of GS/StreamOutput pipeline statistics counters associated with the GS stage and GS threads. This subsection describes these counters and controls depending on device, even in the cases where functions outside of the GS stage (e.g., DataPort) are involved in the statistics gathering.

Refer to the *Statistics Gathering* summary provided earlier in this specification. Refer to the *Memory Interface Registers* chapter for details on these MMIO pipeline statistics counter registers, as well as the chapters corresponding to the other functions involved (e.g., DataPort, URB shared functions).

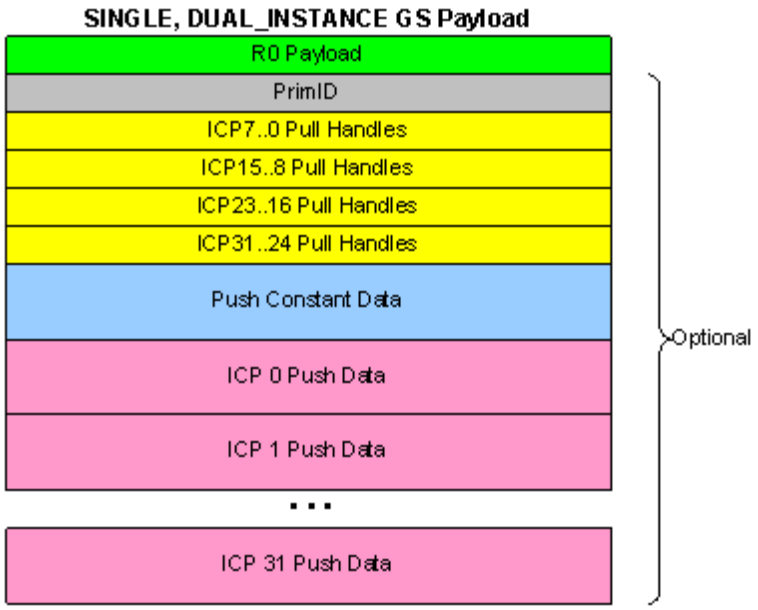
Payloads

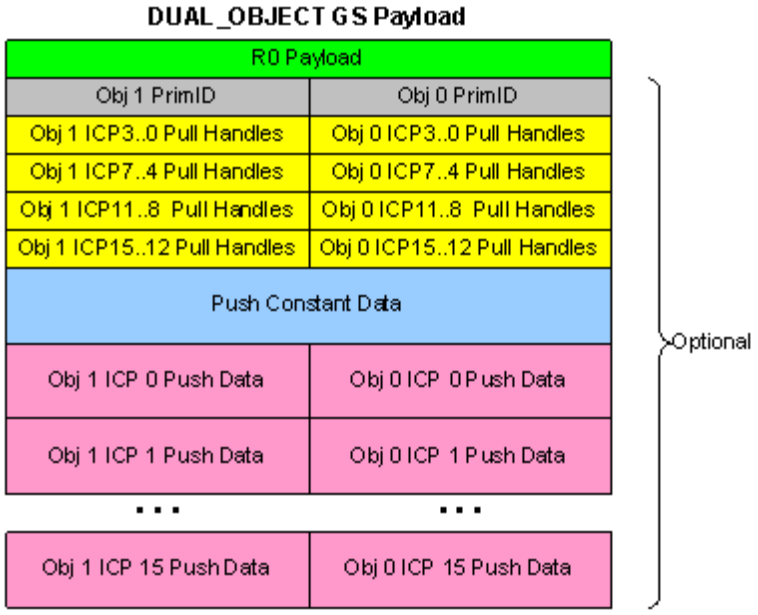
This topic is currently under development.

Thread Payload High-Level Layout

Thread Payload High-Level Layout shows the high-level layout of the payload delivered to GS threads.

GS Dispatch Layouts





Subsequent sections provide detailed layouts for different processor generations.

SIMD 4x2 Thread Payload

The table below shows the layout of the payload delivered to GS threads.

Refer to 3D Pipeline Stage Overview for details on fields that are common among the various pipeline stages.

GRF DWord	Bits	Description
R0.7	31	Reserved
	30:0	Reserved.
R0.6	31	**Dereference Thread.** This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer.** Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9.0	Reserved
	8:0	**FFTID.** This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved.
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table used by this thread, specified as a 32-byte granular offset from the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved.
	3:0	**Per Thread Scratch Space.**Specifies the amount of scratch space allowed for this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). Programming Notes: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:24	**Semaphore Index.** This is a DWord index used in URB_ATOMIC commands if the thread is using data pulled from input handles. This information is only required for pull-model vertex inputs and InstanceCount > 1. Format = U8
	23	Reserved.
	22	**Hint.** This is a copy of the corresponding 3DSTATE_GS bit. Format: U1

GRF DWord	Bits	Description
	21:16	**Primitive Topology Type.** This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV. If the **Discard Adjacency** bit is set, the topology type passed in the payload is UNDEFINED. Format: See *3D Pipeline*.
	15:13	Reserved
	12:0	**Semaphore Handle.** This is the URB offset pointing to the first GS semaphore DWord in the URB. Software is responsible for statically allocating the semaphore DWords in the URB. Refer to Semaphore Handle field in 3DSTATE_GS for size of semaphore allocation.
R0.1	31:27	**GS Instance ID 1.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. If "dispatch mode" is DUAL_OBJECT this field is not valid. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 1.** This is the URB offset where the EU's upper channels (DWords 7:4) results are stored. If only one object/instance is processed (shaded) by the thread, this field is effectively ignored (no results are stored for these channels, as controlled by the thread's Channel Mask).
R0.0	31:27	**GS Instance ID 0.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. If "dispatch mode" is DUAL_OBJECT, this field is not valid. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 0.** This is the URB offset where the EU's lower channels (DWords 3:0) results are stored.
The following register is included only if Include PrimitiveID is enabled.		
R1.7-R1.5	31:0	Reserved: MBZ.
R1.4	31:0	**Primitive ID 1**. This field contains the Primitive ID associated with (all instances) of input object 1. Only valid in DUAL_OBJECT mode. Format: U32
R1.3-R1.1	31:0	Reserved: MBZ.
R1.0	31:0	**Primitive ID 0.** This field contains the Primitive ID associated with (all instances) of input object 0. Format: U32
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled.		
Rn.7	31:16	Description: Reserved
	15:0	ICP 7 Handle

GRF DWord	Bits	Description
Rn.6	31:16	Description: Reserved
	15:0	ICP 6 Handle
Rn.5	31:16	Description: Reserved
	15:0	ICP 5 Handle
Rn.4	31:16	Description: Reserved
	15:0	ICP 4 Handle
Rn.3	31:16	Description: Reserved
	15:0	ICP 3 Handle
Rn.2	31:16	Description: Reserved
	15:0	ICP 2 Handle
Rn.1	31:16	Description: Reserved
	15:0	ICP 1 Handle
Rn.0	31:16	Description: Reserved
	15:0	ICP 0 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+1.7	31:16	Description: Reserved
	15:0	ICP 15 Handle
Rn+1.6	31:16	Description: Reserved
	15:0	ICP 14 Handle
Rn+1.5	31:16	Description: Reserved
	15:0	ICP 13 Handle

GRF DWord	Bits	Description
Rn+1.4	31:16	Description: Reserved
	15:0	ICP 12 Handle
Rn+1.3	31:16	Description: Reserved
	15:0	ICP 11 Handle
Rn+1.2	31:16	Description: Reserved
	15:0	ICP 10 Handle
Rn+1.1	31:16	Description: Reserved
	15:0	ICP 9 Handle
Rn+1.0	31:16	Description: Reserved
	15:0	ICP 8 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 15.		
Rn+2.7	31:16	Description: Reserved
	15:0	ICP 23 Handle
Rn+2.6	31:16	Description: Reserved
	15:0	ICP 22 Handle
Rn+2.5	31:16	Description: Reserved
	15:0	ICP 21 Handle
Rn+2.4	31:16	Description: Reserved
	15:0	ICP 20 Handle
Rn+2.3	31:16	Description: Reserved
	15:0	ICP 19 Handle

GRF DWord	Bits	Description
Rn+2.2	31:16	Description: Reserved
	15:0	ICP 18 Handle
Rn+2.1	31:16	Description: Reserved
	15:0	ICP 17 Handle
Rn+2.0	31:16	Description: Reserved
	15:0	ICP 16 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 23.		
Rn+3.7	31:16	Description: Reserved
	15:0	ICP 31 Handle
Rn+3.6	31:16	Description: Reserved
	15:0	ICP 30 Handle
Rn+3.5	31:16	Description: Reserved
	15:0	ICP 29 Handle
Rn+3.4	31:16	Description: Reserved
	15:0	ICP 28 Handle
Rn+3.3	31:16	Description: Reserved
	15:0	ICP 27 Handle
Rn+3.2	31:16	Description: Reserved
	15:0	ICP 26 Handle
Rn+3.1	31:16	Description: Reserved
	15:0	ICP 25 Handle

GRF DWord	Bits	Description
Rn+3.0	31:16	Description: Reserved
	15:0	ICP 24 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled.		
Rn.7	31:16	Description: Reserved
	15:0	Object 1 ICP 3 Handle
Rn.6	31:16	Description: Reserved
	15:0	Object 1 ICP 2 Handle
Rn.5	31:16	Description: Reserved
	15:0	Object 1 ICP 1 Handle
Rn.4	31:16	Description: Reserved
	15:0	Object 1 ICP 0 Handle
Rn.3	31:16	Description: Reserved
	15:0	Object 0 ICP 3 Handle
Rn.2	31:16	Description: Reserved
	15:0	Object 0 ICP 2 Handle
Rn.1	31:16	Description: Reserved
	15:0	Object 0 ICP 1 Handle
Rn.0	31:16	Description: Reserved
	15:0	Object 0 ICP 0 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 3.		
Rn+1.7	31:16	Description: Reserved
	15:0	Object 1 ICP 7 Handle

GRF DWord	Bits	Description
Rn+1.6	31:16	Description Reserved
	15:0	Object 1 ICP 6 Handle
Rn+1.5	31:16	Description Reserved
	15:0	Object 1 ICP 5 Handle
Rn+1.4	31:16	Description Reserved
	15:0	Object 1 ICP 4 Handle
Rn+1.3	31:16	Description Reserved
	15:0	Object 0 ICP 7 Handle
Rn+1.2	31:16	Description Reserved
	15:0	Object 0 ICP 6 Handle
Rn+1.1	31:16	Description Reserved
	15:0	Object 0 ICP 5 Handle
Rn+1.0	31:16	Description Reserved
	15:0	Object 0 ICP 4 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+2.7	31:16	Description Reserved
	15:0	Object 1 ICP 11 Handle
Rn+2.6	31:16	Description Reserved
	15:0	Object 1 ICP 10 Handle
Rn+2.5	31:16	Description Reserved
	15:0	Object 1 ICP 9 Handle

GRF DWord	Bits	Description
Rn+2.4	31:16	Description: Reserved
	15:0	Object 1 ICP 8 Handle
Rn+2.3	31:16	Description: Reserved
	15:0	Object 0 ICP 11 Handle
Rn+2.2	31:16	Description: Reserved
	15:0	Object 0 ICP 10 Handle
Rn+2.1	31:16	Description: Reserved
	15:0	Object 0 ICP 9 Handle
Rn+2.0	31:16	Description: Reserved
	15:0	Object 0 ICP 8 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 11.		
Rn+3.7	31:16	Description: Reserved
	15:0	Object 1 ICP 15 Handle
Rn+3.6	31:16	Description: Reserved
	15:0	Object 1 ICP 14 Handle
Rn+3.5	31:16	Description: Reserved
	15:0	Object 1 ICP 13 Handle
Rn+3.4	31:16	Description: Reserved
	15:0	Object 1 ICP 12 Handle
Rn+3.3	31:16	Description: Reserved
	15:0	Object 0 ICP 15 Handle

GRF DWord	Bits	Description
Rn+3.2	31:16	Description Reserved
	15:0	Object 0 ICP 14 Handle
Rn+3.1	31:16	Description Reserved
	15:0	Object 0 ICP 13 Handle
Rn+3.0	31:16	Description Reserved
	15:0	Object 0 ICP 12 Handle
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
Varies	31:0	**Pushed Vertex Data.** There can be up to 32 vertices supplied, each with a size defined by the **Vertex URB Entry Read Length** state. The amount of data provided for each vertex is defined by the **Vertex URB Entry Read Length** state. For SINGLE or DUAL_INSTANCE dispatch modes, the pushed data for Vertex 0 immediately follows any pushed constant data. The pushed data for Vertex 1 immediately follows Vertex 0, and so on. There is no upper/lower swizzling of data. For DUAL_OBJECT dispatch mode, the pushed vertex data is split into upper and lower halves with Object 0 input vertices in the lower half, and Object 1 input vertices in the upper half.

SIMD8 Thread Payload

The table below shows the layout of the payload delivered to GS threads for SKL+.

Refer to the 3D Pipeline Stage Overview section in *3D Pipeline* for details on those fields that are common among the various pipeline stages.

GRF DWord	Bits	Description
R0.7	31	Reserved
	30:0	Reserved.
R0.6	31:24	Reserved.
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer.** Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address.

GRF DWord	Bits	Description
		Format = GeneralStateOffset[31:10]
	9:0	**FFTID.** This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for Implementation Use
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	Reserved.
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved.
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). (See Generic Pipeline Stage for further description). **Programming Notes:** This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:23	Reserved.
	22	**Hint:** This is a copy of the corresponding 3DSTATE_GS bit. Format: U1
	21:16	**Primitive Topology Type.** This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV. If the **Discard Adjacency** bit is set, the topology type passed in the payload is UNDEFINED. Format: See 3D Pipeline
	15:0	Reserved.
R0.1-R0.0	31:0	Reserved.

GRF DWord	Bits	Description
R1.7	31:0	GS Instance ID / URB Return Handle for Object 7 (See R1.0)
R1.6	31:0	GS Instance ID / URB Return Handle for Object 6 (See R1.0)
R1.5	31:0	GS Instance ID / URB Return Handle for Object 5 (See R1.0)
R1.4	31:0	GS Instance ID / URB Return Handle for Object 4 (See R1.0)
R1.3	31:0	GS Instance ID / URB Return Handle for Object 3 (See R1.0)
R1.2	31:0	GS Instance ID / URB Return Handle for Object 2 (See R1.0)
R1.1	31:0	GS Instance ID / URB Return Handle for Object 1 (See R1.0)
R1.0	31:27	**GS Instance ID 0.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 0.** This is the URB offset where the EU's lower channels (DWords 3:0) results are to be stored. Format: U14 64B-aligned URB Offset
The following register is included only if **Include PrimitiveID** is enabled.		
R2.7	31:0	**Primitive ID 7.** This field contains the Primitive ID associated with input object 7 (or the single input object if InstanceCount>1) Format: U32
R2.6	31:0	**Primitive ID 6.** This field contains the Primitive ID associated with input object 6 (or the single input object if InstanceCount>1) Format: U32
R2.5	31:0	**Primitive ID 5.** This field contains the Primitive ID associated with input object 5 (or the single input object if InstanceCount>1) Format: U32
R2.4	31:0	**Primitive ID 4.** This field contains the Primitive ID associated with input object 4 (or the single input object if InstanceCount>1) Format: U32
R2.3	31:0	**Primitive ID 3.** This field contains the Primitive ID associated with input object 3 (or the single input object if InstanceCount>1) Format: U32

GRF DWord	Bits	Description
R2.2	31:0	**Primitive ID 2.** This field contains the Primitive ID associated with input object 2 (or the single input object if InstanceCount>1) Format: U32
R2.1	31:0	**Primitive ID 1.** This field contains the Primitive ID associated with input object 1 (or the single input object if InstanceCount>1) Format: U32
R2.0	31:0	**Primitive ID 0.** This field contains the Primitive ID associated with input object 0 (or the single input object if InstanceCount>1) Format: U32
The following registers are included only if **Include Vertex Handles** is enabled and **InstanceCount** == 1.		
Rn.7	31:16	Reserved.
	15:0	Object 7 ICP 0 Handle
Rn.6	31:16	Reserved.
	15:0	Object 6 ICP 0 Handle
Rn.5	31:16	Reserved.
	15:0	Object 5 ICP 0 Handle
Rn.4	31:16	Reserved.
	15:0	Object 4 ICP 0 Handle
Rn.3	31:16	Reserved.
	15:0	Object 3 ICP 0 Handle
Rn.2	31:16	Reserved.
	15:0	Object 2 ICP 0 Handle
Rn.1	31:16	Reserved.
	15:0	Object 1 ICP 0 Handle
Rn.0	31:16	Reserved.
	15:0	Object 0 ICP 0 Handle
[Rn+1]	255:0	ICP 1 Handle for Objects 0-7
[Rn+2]	255:0	ICP 2 Handle for Objects 0-7
...		...
[Rn+32]	255:0	ICP 32 Handle for Objects 0-7
The following registers are included only if **Include Vertex Handles** is enabled and **InstanceCount** > 1.		
Rn.7	31:16	Reserved.
	15:0	ICP 7 Handle (if required)

GRF DWord	Bits	Description
Rn.6	31:16	Reserved.
	15:0	ICP 6 Handle (if required)
Rn.5	31:16	Reserved.
	15:0	ICP 5 Handle (if required)
Rn.4	31:16	Reserved.
	15:0	ICP 4 Handle (if required)
Rn.3	31:16	Reserved.
	15:0	ICP 3 Handle (if required)
Rn.2	31:16	Reserved.
	15:0	ICP 2 Handle (if required)
Rn.1	31:16	Reserved.
	15:0	ICP 1 Handle (if required)
Rn.0	31:16	Reserved.
	15:0	ICP 0 Handle
[Rn+1]	255:0	ICP 8-15 Handle
[Rn+2]	255:0	ICP 16-23 Handle
[Rn+3]	255:0	ICP 24-31 Handle
[Varies] optional	255:0	**Constant Data (optional):** Please refer to the Push Constants chapter in the General Programming of Thread-Generating Stages section for more details on size and source of constant data.
Varies		**Pushed Vertex Data (InstanceCount == 1 Case): (optional)** Input data for the 8 input objects is located here. Object 0 (starting with Vertex 0 of Object 0) data is passed in DW0 of these GRFs, and Object 7 data is passed in DW7. The first GRF contains Vertex 0 Element 0 Component 0 for all 8 objects, followed by components 1-3 in the three subsequent GRFs. This is followed by GRFs containing Vertex 0 Element 1 (if it exists), and so on, up to the number of Vertex 0 elements specified by **Vertex URB Read Length**. This is followed by the data for Vertex 1 for all objects (if it exists), and so on until all relevant vertices are passed. Note that the amount of data passed is limited by the number of GRFs supported by EUs. Software is responsible for comprehending this limit and resorting to the pull model as required.
Rv.7	31:0	**Object 7 Vertex 0 Element 0 Component 0**
Rv.6	31:0	**Object 6 Vertex 0 Element 0 Component 0**
Rv.5	31:0	**Object 5 Vertex 0 Element 0 Component 0**
Rv.4	31:0	**Object 4 Vertex 0 Element 0 Component 0**
Rv.3	31:0	**Object 3 Vertex 0 Element 0 Component 0**
Rv.2	31:0	**Object 2 Vertex 0 Element 0 Component 0**

GRF DWord	Bits	Description
Rv.1	31:0	**Object 1 Vertex 0 Element 0 Component 0**
Rv.0	31:0	**Object 0 Vertex 0 Element 0 Component 0**
Rv+1	31:0	**Object 0-7 Vertex 0 Element 0 Component 1**
...		and so on...
Varies		**Pushed Vertex Data (InstanceCount > 1 Case): (optional)** Input data for the single input object (shared across all instances) is located here. The pushed data for Vertex 0 immediately follows any pushed constant data. The pushed data for Vertex 1 immediately follows Vertex 0, and so on. There is no upper/lower swizzling of data.

Stream Output Logic (SOL) Stage

The Stream Output Logic (SOL) stage receives 3D topologies originating in the VF, DS or GS stage. If enabled, the SOL stage uses programmed state information to copy portions of the vertex data associated with the incoming topologies across one or more Stream Output (SO) Buffers.

State

This section contains state commands and structures pertaining to the StreamOut Logic (SOL) stage of the 3D pipeline.

3DSTATE_STREAMOUT

The 3DSTATE_STREAMOUT command specifies control information for the SOL stage. Included are enables and sizes for input streams and enables for output buffers.

The SOL unit incorrectly double buffers MMIO/NP registers and only moves them into the design for usage when control topology is received with the SOL unit dirty state.

If the state does not change, need to resend the same state.

There is no need to send a pipeline state update to the SOL unit after SOL unit MMIO registers or non-pipeline state are written.

3DSTATE_STREAMOUT

3DSTATE_SO_DECL_LIST Command

The 3DSTATE_SO_DECL_LIST instruction defines a list of Stream Output (SO) declaration entries (SO_DECLs) and associated information for all specific SO streams in parallel.

3DSTATE_SO_DECL_LIST

SO_DECL

3DSTATE_SO_BUFFER

The 3DSTATE_SO_BUFFER command specifies the location and characteristics of an SO buffer in memory.

Command
3DSTATE_SO_BUFFER

The SOL Unit also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_WM, 3DSTATE_PS_EXTRA, and 3DSTATE_DEPTH_STENCIL_STATE fields.

Signal	Description	Formula
SOL_INT::Render_Enable	If clear, the SO stage will not forward any topologies down the pipeline. If set, the SO stage will forward topologies associated with Render Stream Select down the pipeline. This bit is used even if SO Function Enable is DISABLED.	= (3DSTATE_STREAMOUT::Force_Rending == Force_On) || ((3DSTATE_STREAMOUT::Force_Rending != Force_Off) && !(3DSTATE_GS::Enable && 3DSTATE_GS::Output Vertex Size == 0) & !3DSTATE_STREAMOUT::API_Render_Disable && (3DSTATE_DEPTH_STENCIL_STATE::Stencil_TestEnable || 3DSTATE_DEPTH_STENCIL_STATE::Depth_TestEnable || 3DSTATE_DEPTH_STENCIL_STATE::Depth_WriteEnable || 3DSTATE_PS_EXTRA::PS_Valid || 3DSTATE_WM::Legacy Depth_Buffer_Clear || 3DSTATE_WM::Legacy Depth_Buffer_Resolve_Enable || 3DSTATE_WM::Legacy Hierarchical_Depth_Buffer_Resolve_Enable))

DW1[21]	DW1[20]	Stream Offset	Action
Full legacy mode. HW doesn't LOAD or STORE, it simply updates the MMIO register during stream out. SW can can the LOAD/STORE using MI_LOAD_REG/ MI_STORE_REG.			
0	0	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = no action
0	0	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = no action
SW can cause the LOAD of the SO_OFFSET using MI_LOAD_REG, and HW performs the STORE.			
0	1	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = No action, write SO_WRITE_OFFSET[x] to memory
0	1	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = No action, write SO_WRITE_OFFSET[x] to memory
HW performs the LOAD, and SW can cause the STOREs using MI_STORE_REG_MEM.			
1	0	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = stream offset
1	0	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = load from memory
HW performs both the LOAD and STORE.			
1	1	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = stream offset, write SO_WRITE_OFFSET[x] to memory
1	1	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = load from memory, write SO_WRITE_OFFSET[x] to memory

"SO_WRITE_OFFSET[x] =" occurs before the execution of the primitive, while write SO_WRITE_OFFSET[x] to memory occurs after the execution of the primitive.

Functions

Input Buffering

For the purposes of stream output, the SOL stage breaks incoming topologies into independent objects without adjacency information. In the process, any adjacent-only vertices are ignored. For example, it converts TRISTRIP_ADJ into independent 3-vertex triangles. However, if rendering is enabled, incoming topologies are passed to the Clip stage unmodified and therefore the Clip unit must be enabled if there is any possibility of "ADJ" topologies reaching it.

Note that the SOL unit will not see incomplete objects: the VF will remove incomplete input objects, the GS will remove GS-generated incomplete objects, and the DS does not output incomplete objects as only complete topologies are generated by the TE stage.

The OSB (Object Staging Buffer) reorders the vertices of odd-numbered triangles in TRISTRIP topologies to match API requirements.

Incoming topologies are tagged with a 2-bit StreamID. The StreamID is 0 for topologies originating from the VF stage (i.e., 3DPRIMITIVE_xxx). For topologies output from the GS stage, the StreamID is set by the GS shader. A Stream *n* Vertex Length is associated with each stream, and defines how much data is read from the URB for vertices in that stream.

The following table specifies how the SOL stage streams out object vertices for each incoming topology type.

PrimTopologyType	Order of Vertices Streamed Out	
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...);	Any SOL Notes
POINTLIST POINTLIST_BF	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
LINELIST (N is multiple of 2)	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
LINELIST_ADJ (N is multiple of 4)	[0] = (1,2); [1] = (5,6); ...; [(N/4)-1)] = (N-3,N-2)	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF (N >= 2)	[0] = (0,1); [1] = (1,2); ...; [N-2] = (N-2,N-1)	
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT (N >= 4)	[0] = (1,2); [1] = (2,3); ...; [N-4] = (N-3,N-2)	LINESTRIP_ADJ_CONT is added for BDW+. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINELOOP	N/A	Not supported after VF.
TRILIST (N is multiple of 3)	[0] = (0,1,2); [1] = (3,4,5); ...; [(N/3)-1] = (N-3,N-2,N-1)	
RECTLIST, RECTLIST_SUBPIXEL	Same as TRILIST	Handled same as TRILIST.
TRILIST_ADJ (N is multiple of 6)	[0] = (0,2,4); [1] = (6,8,10); ...; [(N/6)-1] = (N-6,N-4,N-2)	
TRISTRIP (N >= 3) REORDER_LEADING	[0] = (0,1,2); [1] = (1,3,2); [k even] = (k,k+1,k+2) [k odd] = (k,k+2,k+1) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP (N >= 3) REORDER_TRAILING	[0] = (0,1,2); [1] = (2,1,3); [k even] = (k,k+1,k+2) [k odd] = (k+1,k,k+2) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing trailing vertices starting with v2.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
TRISTRIP_REV (N >= 3) REORDER_LEADING	[0] = (0,2,1) [1] = (1,2,3);...; [k even] = (k,k+2,k+1) [k odd] = (k,k+1,k+2) [N-3] = (see above)	"Even" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP_REV (N >= 3) REORDER_TRAILING	[0] = (1,0,2) [1] = (1,2,3);...; [k even] = (k+1,k,k+2) [k odd] = (k,k+1,k+2) [N-3] = (see above)	"Even" triangles have vertices reordered to yield increasing trailing vertices starting with v2.
TRISTRIP_ADJ (N even, N >= 6) REORDER_LEADING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (2,6,4); ...; N > 10: [0] = (0,2,4); [1] = (2,6,4); ...; [k>1, even] = (2k, 2k+2, 2k+4); [k>2, odd] = (2k, 2k+4, 2k+2);...; Trailing object: [(N/2)-3, even] = (N-6,N-4,N-2); [(N/2)-3, odd] = (N-6,N-2,N-4);	"Odd" objects have vertices reordered to yield increasing-by-2 leading vertices starting with v0.
TRISTRIP_ADJ (N even, N >= 6) REORDER_TRAILING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (4,2,6); ...; N > 10: [0] = (0,2,4); [1] = (4,2,6); ...; [k>1, even] = (2k,	"Odd" objects have vertices reordered to yield increasing-by-2 trailing vertices starting with v4.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
	2k+2, 2k+4); [k>2, odd] = (2k+2,2k, 2k+4,);...; Trailing object: [(N/2)-3, even] = (N-6,N-4,N-2); [(N/2)-3, odd] = (N-4,N-6,N-2);	
TRIFAN (N > 2)	[0] = (0,1,2); [1] = (0,2,3); ...; [N-3] = (0, N-2, N-1);	
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON, POLYGON_CONT	Same as TRIFAN	POLYGON_CONT is added for BDW+. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
QUADLIST QUADSTRIP	N/A	Not supported after VF.
PATCHLIST_1	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
PATCHLIST_2	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
PATCHLIST_3..32	similar to above	

Stream Output Function

As previously mentioned, incoming 3D topologies are targeted at one of the four streams. The SOL stage contains state information specific to each of the four streams.

A stream's list of SO declarations (SO_DECL structures) is used to perform the SO function for objects targeted to that particular stream. The 3DSTATE_SO_DECL_LIST command is used to specify the list of SO_DECL structures for all four streams in parallel. Software is required to scan the SO_DECL lists of streams to determine which SO buffers are targeted. The Stream To Buffer Selects bits in 3DSTATE_SO_DECL_LIST must be programmed accordingly (if the buffer is targeted, the select bit must set, else it must be cleared).

If a stream has no SO_DECL state defined (NumEntries is 0), incoming objects targeting that stream are effectively ignored. As there is no attempt to perform stream output, overflow detection is neither required nor performed.

Otherwise, an overflow check is performed. First any attempt to output to a disabled buffer is detected. This occurs when the stream has a Stream To Buffer Selects bit set but the corresponding SO Buffer Enable is clear. Assuming all targeted buffers are enabled, an additional check is made to ensure that there is enough room in each targeted buffer to hold the number of vertices which be output to it (for the input object). Here the buffer's current end address is compared to what the write offset would be if the output was performed. The latter value is computed as (write_offset + vertex_count * buffer_pitch). If this value is greater than the end address, an overflow is signalled. This check is performed for each buffer included in Stream To Buffer Selects.

If an overflow is not signaled, the SO function is performed. The SO_DECL list for the targeted stream is traversed independently for each object vertex, and the operation specified by the SO_DECL structure is performed (typically causing data to be appended to an SO buffer). In the process, SO buffer Write Offsets are incremented.

Stream Output Buffers

Up to four SO buffers are supported. The SO buffer parameters (start/end address, etc.) are specified by the 3DSTATE_SO_BUFFER command.

The 3DSTATE_STREAMOUT command specifies an SO Buffer Enable bit for each of the buffers. If a buffer is disabled, its state is ignored and no output will be attempted for that buffer. Any attempt to output to that buffer will immediately signal an overflow condition.

The SOL stage maintains a current Write Offset register value for each SO buffer. These registers can be written via MI_LOAD_REGISTER_MEM or MI_LOAD_REGISTER_IMM commands. The SOL stage will increment the Write Offsets as a part of the SO function. Software can cause a Write Offset register to be written to memory via an MI_STORE_REGISTER_MEM command, though a preceding flush operation may be required to ensure that any previous SO functions have completed.

Surface Format Name
R32G32B32A32_FLOAT
R32G32B32A32_SINT
R32G32B32A32_UINT
R32G32B32_FLOAT
R32G32B32_SINT
R32G32B32_UINT
R32G32_FLOAT
R32G32_SINT
R32G32_UINT
R32_SINT

Surface Format Name
R32_UINT
R32_FLOAT

Rendering Disable

Independent of SOL function enable, if rendering (i.e, 3D pipeline functions past the SOL stage) is enabled (via clearing the Rendering Disable bit), the SOL stage will pass topologies for a specific input stream (as selected by Render Stream Select) down the pipeline, with the exception of PATCHLIST_n topologies which are never passed downstream. Software must ensure that the vertices exiting the SOL stage include a vertex header and position value so that the topologies can be correctly processed by subsequent pipeline stages. Specifically, rendering must be disabled whenever 128-bit vertices are output from a GS thread.

If Rendering Disable is set, the SOL stage will prevent any topologies from exiting the SOL stage.

Statistics

The SOL stage controls the incrementing of two 64-bit statistics counter registers for each of the four output buffer slots, SO_NUM_PRIMS_WRITTEN[] and SO_PRIM_STORAGE_NEEDED[].

3D Pipeline Rasterization

Common Rasterization State

This section contains rasterization state pointers.

3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_RASTER

3D Pipeline – CLIP Stage Overview

The CLIP stage of the GEN 3D Pipeline is similar to the GS stage in that it can be used to perform general processing on incoming 3D objects via spawned GEN4 threads. However, the CLIP stage also includes specialized logic to perform a *ClipTest* function on incoming objects. These two usage models of the CLIP stage are outlined below.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D Pipeline stage, as much of the CLIP stage operation and control falls under these "common" functions. I.e., many of the CLIP stage state variables and CLIP thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the CLIP stage, details on the ClipTest function, and any exceptions the CLIP stage exhibits with respect to common FF unit functions.

Clip Stage – 3D Clipping

The ClipTest fixed function is provided to optimize the CLIP stage for support of generalized *3D Clipping*. The CLIP FF unit examines the position of incoming vertices, performs a fixed function *VertexClipTest* on these positions, and then examines the results for the vertices of each independent object in *ClipDetermination*.

The results of ClipDetermination indicate whether an object is to be clipped (MustClip), discarded (TrivialReject) or passed down the pipeline unmodified (TrivialAccept). In the MustClip case, the fixed function clipping hardware is responsible for performing the actual 3D Clipping algorithm. The CLIP hardware is passed the source object vertex data and is able to output a new, arbitrary 3D primitive (e.g., the clipped primitive), or no output at all. Note that the output primitive is comprised of newly-generated vertex positions, barycentric attributes and shares vertices with the source primitive for rest of the attributes.The CLIP unit maintains the proper ordering of CLIP-generated primitives and any surrounding trivially-accepted primitives and processes all the primitives in order.

The outgoing primitive stream is sent down the pipeline to the Strip/Fan (SF) FF stage (now including the read-back VUE Vertex Header data such as Vertex Position (NDC or screen space), RTAIndex, VPIndex, PointWidth) and control information (PrimType, PrimStart, PrimEnd) while the remainder of the vertex data remains in the VUE in the URB.

Fixed Function Clipper

The GPU supports Fixed Function Clipping.

Note: In an earlier generation, clipping was done in the EU. However the clipper thread latency was high and caused a bottleneck in the pipeline. Hence the motivation for a fixed function clipper.

Concepts

This section provides an overview of 3D clip-testing and clipping concepts, as defined by the Direct3D* and OpenGL* APIs. It is provided as background material. Some of the concepts impact HW functionality while others impact CLIP kernel functionality.

* Other names and brands may be claimed as the property of others.

CLIP Stage Input

As a stage of the GEN 3D pipeline, the CLIP stage receives inputs from the previous (GS) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the CLIP stage.

State

This section contains state clips for the Clip Stage. For each processor generation, the state used by the clip stage is defined by the appropriate inline state packet, linked below.

3DSTATE_CLIP

3D_STATE_CLIP

The Clip unit will unconditionally reject incoming PATCHLIST topologies, if not already discarded by SOL. So there is no need for SW to explicitly set the CLIP_mode to reject PATCHLIST topologies.

Clip Unit also receives 3DSTATE_RASTER. It also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_VS, 3DSTATE_DS and 3DSTATE_GS fields.

Signal	Description	Formula
CLIP_INT::Front_Winding	Determines whether a triangle object is considered “front facing” if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.	= 3DSTATE_RASTER::FrontWinding
CLIP_INT::CullMode	Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points, or rectangles.	= 3DSTATE_RASTER::CullMode
CLIP_INT::Viewport Z ClipTest Enable	This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.	= 3DSTATE_RASTER::Viewport Z ClipTest Enable
CLIP_INT::User Clip Distance Cull Test Enable Bitmask	This 8-bit mask field selects which of the 8 user clip distances against which trivial reject/trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances. This mask must be mutually exclusive to final CLIP_INT::User Clip Distance Clip Test Enable Bitmask. Same mask bit can't be set	= (3DSTATE_CLIP::ForceUser Clip Distance Cull Test Enable Bitmask == Force) ? 3DSTATE_CLIP::User Clip Distance Cull Test Enable Bitmask : 3DSTATE_GS::GS_Enable ? 3DSTATE_GS:: GS User Clip Distance Cull Test Enable Bitmask : 3DSTATE_DS:DS_Enable ? 3DSTATE_DS:: DS User Clip Distance Cull Test Enable

Signal	Description	Formula
	for both.	Bitmask : 3DSTATE_INT:VS_Enable ? 3DSTATE_VS:: VS User Clip Distance Cull Test Enable Bitmask : 0
CLIP_INT::User Clip Distance Clip Test Enable Bitmask	This 8-bit mask field selects which of the 8 user clip distances against which trivial reject/trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Clip Distance test of up to 8 distances. This mask must be mutually exclusive to final CLIP_INT::User Clip Distance Cull Test Enable Bitmask. Same mask bit can't be set for both.	= (3DSTATE_CLIP::ForceUser Clip Distance Clip Test Enable Bitmask == Force) ? 3DSTATE_CLIP::User Clip Distance Clip Test Enable Bitmask : 3DSTATE_GS:GS_Enable ? 3DSTATE_GS::GS User Clip Distance Clip Test Enable Bitmask : 3DSTATE_DS::DS_Enable ? 3DSTATE_DS::DS User Clip Distance Clip Test Enable Bitmask : 3DSTATE_VS::VS_Enable ? 3DSTATE_VS::VS User Clip Distance Clip Test Enable Bitmask : 0

VUE Readback

Starting with the CLIP stage, the 3D pipeline requires vertex information in addition to the VUE handle. For example, the CLIP unit's VertexClipTest function needs the vertex position, as does the SF unit's functions. This information is obtained by the 3D pipeline reading a portion of each vertex's VUE data directly from the URB. This readback (effectively) occurs immediately before the CLIP VertexClipTest function, and immediately after a CLIP thread completes the output of a destination VUE.

The Vertex Header (first 256 bits) of the VUE data is read back. (See the previous *VUE Formats* subsection (above) for details on the content and format of the Vertex Header.) Additional Clip/Cull data (located immediately past the Vertex Header) may be read prior to clipping.

This readback occurs automatically and is not under software control. The only software implication is that the Vertex Header must be valid at the readback points, and therefore must have been previously loaded or written by a thread.

VertexClipTest Function

The VertexClipTest function compares each incoming vertex position (x,y,z,w) with various viewport and guardband parameters (either hard-coded values or specified by state variables).

The RHW component of the incoming vertex position is tested for NaN value, and if a NaN is detected, the vertex is marked as "BAD" by setting the outcode[BAD]. If a NaN is detected in any vertex homogeneous x,y,z,w component or an enabled ClipDistance value, the vertex is marked as "BAD" by setting the outcode[BAD].

In general, any object containing a BAD vertex will be discarded, as how to clip/render such objects is undefined.

However, in the case of CLIP_ALL mode, a CLIP thread will be spawned even for objects with "BAD" vertices. The CLIP kernel is required to handle this case, and can examine the **Object Outcode [BAD]** payload bit to detect the condition. (Note that the VP and GB Object Outcodes are UNDEFINED when BAD is set.)

If the incoming RHW coordinate is negative (including negative 0) the NEGW outcode is set. Also, this condition is used to select the proper comparison functions for the VP and GB outcode tests (below).

Next, the VPXY and GB outcodes are computed, depending on the corresponding enable SV bits. If one of VPXY or GB is disabled, the enabled set of outcodes are copied to the disabled set of outcodes. This effectively defines the disabled boundaries to coincide with the enabled boundaries (i.e., disabling the GB is just like setting it to the VPXY values, and vice versa).

The VPZ outcode is computed as required by the API mode SV.

Separate VP Z Near (ZMin) and Z Far (ZMax) ClipTest Enable state bits are provided in 3DSTATE_RASTER.

Finally, the incoming UserClipFlags are masked and copied to corresponding outcodes.

The following algorithm is used by VertexClipTest:

```
//
// Vertex ClipTest
//
// On input:
// if (CLIP.PreMapped)
//   x,y are viewport mapped
//   z is NDC ([0,1] is visible)
// else
//   x,y,z are NDC (post-perspective divide)
//   w is always 1/w
//
// Initialize outCodes to "inside"
//
outCode[*] = 0
//
// Check if w is NaN
// Any object containing one of these "bad" vertices will likely be discarded
//
if (ISNAN(homogeneous x,y,z,w or enabled ClipDistance value)
{
   outCode[BAD] = 1
}
```

```
//
// If 1/w is negative, w is negative and therefore outside of the w=0 plane
//
//
rhw_neg = ISNEG(rhw)
if (rhw_neg)
{
   outCode[NEGW] = 1
}
//
// View Volume Clip Test
// If Premapped, the 2D viewport is defined in screen space
// otherwise the canonical NDC viewvolume applies ([-1,1])
//
if (CLIP_STATE.PreMapped)
{
   vp_XMIN = CLIP_STATE.VP_XMIN
   vp_XMAX = CLIP_STATE.VP_XMAX
   vp_YMIN = CLIP_STATE.VP_YMIN
   vp_YMAX = CLIP_STATE.VP_YMAX
} else {
   vp_XMIN = -1.0f
   vp_XMAX = +1.0f
   vp_YMIN = -1.0f
   vp_YMAX = +1.0f
}

if (CLIP_STATE.ViewportXYClipTestEnable) {
   outCode[VP_XMIN] = (x < vp_XMIN)
   outCode[VP_XMAX] = (x > vp_XMAX)
   outCode[VP_YMIN] = (y < vp_YMIN)
   outCode[VP_YMAX] = (y > vp_YMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
      outCode[VP_XMIN] = (x >= vp_XMIN)
      outCode[VP_XMAX] = (x <= vp_XMAX)
      outCode[VP_YMIN] = (y >= vp_XMIN)
      outCode[VP_YMAX] = (y <= vp_XMAX)
   }
   #endif
   if (rhw_neg) {
      outCode[VP_XMIN] = (x > vp_XMIN)
      outCode[VP_XMAX] = (x < vp_XMAX)
      outCode[VP_YMIN] = (y > vp_XMIN)
      outCode[VP_YMAX] = (y < vp_XMAX)
   }
}

if (CLIP_STATE.ViewportZClipTestEnable) {
   if (CLIP_STATE.APIMode == APIMODE_D3D) {
      vp_ZMIN = 0.0f
      vp_ZMAX = 1.0f
   } else { // OGL
      vp_ZMIN = -1.0f
      vp_ZMAX = 1.0f
   }
   outCode[VP_ZMIN] = (z < vp_ZMIN)
   outCode[VP_ZMAX] = (z > vp_ZMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
```

```
      outCode[VP_ZMIN] = (z >= vp_ZMIN)
      outCode[VP_ZMAX] = (z <= vp_ZMAX)
   }
   #endif
   if (rhw_neg) {
      outCode[VP_ZMIN] = (z > vp_ZMIN)
      outCode[VP_ZMAX] = (z < vp_ZMAX)
   }
}
//
// Guardband Clip Test
//
if {CLIP_STATE.GuardbandClipTestEnable) {
   gb_XMIN = CLIP_STATE.Viewport[vpindex].GB_XMIN
   gb_XMAX = CLIP_STATE.Viewport[vpindex].GB_XMAX
   gb_YMIN = CLIP_STATE.Viewport[vpindex].GB_YMIN
   gb_YMAX = CLIP_STATE.Viewport[vpindex].GB_YMAX
   outCode[GB_XMIN] = (x < gb_XMIN)
   outCode[GB_XMAX] = (x > gb_XMAX)
   outCode[GB_YMIN] = (y < gb_YMIN)
   outCode[GB_YMAX] = (y > gb_YMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
      outCode[GB_XMIN] = (x >= gb_XMIN)
      outCode[GB_XMAX] = (x <= gb_XMAX)
      outCode[GB_YMIN] = (y >= gb_YMIN)
      outCode[GB_YMAX] = (y <= gb_YMAX)
   }
   #endif
   if (rhw_neg) {
      outCode[GB_XMIN] = (x > gb_XMIN)
      outCode[GB_XMAX] = (x < gb_XMAX)
      outCode[GB_YMIN] = (y > gb_YMIN)
      outCode[GB_YMAX] = (y < gb_YMAX)
   }
}
//
// Handle case where either VP or GB disabled (but not both)
// In this case, the disabled set take on the outcodes of the enabled set
//
if (CLIP_STATE.ViewportXYClipTestEnable && !CLIP_STATE.GuardbandClipTestEnable) {
   outCode[GB_XMIN] = outCode[VP_XMIN]
   outCode[GB_XMAX] = outCode[VP_XMAX]
   outCode[GB_YMIN] = outCode[VP_YMIN]
   outCode[GB_YMAX] = outCode[VP_YMAX]
} else if (!CLIP_STATE.ViewportXYClipTestEnable && CLIP_STATE.GuardbandClipTestEnable) {
   outCode[VP_XMIN] = outCode[GB_XMIN]
   outCode[VP_XMAX] = outCode[GB_XMAX]
   outCode[VP_YMIN] = outCode[GB_YMIN]
   outCode[VP_YMAX] = outCode[GB_YMAX]
}
//
// X/Y/Z NaN Handling
//
xyorgben = (CLIP_STATE.ViewportXYClipTestEnable || CLIP_STATE.GuardbandClipTestEnable)
if (isNAN(x)) {
   outCode[GB_XMIN] = xyorgben
   outCode[GB_XMAX] = xyorgben
   outCode[VP_XMIN] = xyorgben
   outCode[VP_XMAX] = xyorgben
}
```

```
if (isNAN(y)) {
   outCode[GB_YMIN] = xyorgben
   outCode[GB_YMAX] = xyorgben
   outCode[VP_YMIN] = xyorgben
   outCode[VP_YMAX] = xyorgben
}
if (isNaN) {
   outCode[VP_ZMIN] = CLIP_STATE.ViewportZClipTestEnable
   outCode[VP_ZMAX] = CLIP_STATE.ViewportZClipTestEnable
}
//
// UserClipFlags
//
ExamineUCFs
for (i=0; i<7; i++)
{
   outCode[UC0+i] = userClipFlag[i] & CLIP_STATE.UserClipFlagsClipTestEnableBitmask[i]
}
outCode[UC7] = userClipFlag[i] & CLIP_STATE.UserClipFlagsClipTestEnableBitmask[7]
```

Object Staging

The CLIP unit's Object Staging Buffer (OSB) accepts streams of input vertex information packets, along with each vertex's VertexClipTest result (outCode). This information is buffered until a complete object or the last vertex of the primitive topology is received. The OSB then performs the ClipDetermination function on the object vertices, and takes the actions required by the results of that function.

Partial Object Removal

The OSB is responsible for removing incomplete LINESTRIP and TRISTRIP objects that it may receive from the preceding stage (GS). Partial object removal is not supported for other primitive types due to either (a) the GS is not permitted to output those primitive types (e.g., primitives with adjacency info), and the VF unit will have removed the partial objects as part of 3DPRIMITIVE processing, or (b) although the GS thread is allowed to output the primitive type (e.g., LINELIST), it is assumed that the GS kernel will be correctly implemented to avoid outputting partial objects (or pipeline behavior is UNDEFINED).

An object is considered 'partial' if the last vertex of the primitive topology is encountered (i.e., PrimEnd is set) before a complete set of vertices for that object have been received. Given that only LINESTRIP and TRISTRIP primitive types are subject to CLIP unit partial object removal, the only supported cases of partial objects are 1-vertex LINESTRIPs and 1 or 2-vertex TRISTRIPs.

ClipDetermination Function

In ClipDetermination, the vertex outcodes of the primitive are combined in order to determine the clip status of the object (TR: trivially reject; TA: trivial accept; MC: must clip; BAD: invalid coordinate). Only those vertices included in the object are examined (3 vertices for a triangle, 2 for a line, and 1 for a point). The outcode bit arrays for the vertices are separately ANDed (intersection) and ORed (union) together (across vertices, not within the array) to yield objANDCode and objORCode bit arrays.

TR/TA against interesting boundary subsets are then computed. The TR status is computed as the logical OR of the appropriate objANDCode bits, as the vertices need only be outside of one common boundary

to be trivially rejected. The TA status is computed as the logical NOR of the appropriate objORCode bits, as any vertex being outside of any of the boundaries prevents the object from being trivially accepted.

If any vertex contains a BAD coordinate, the object is considered BAD and any computed TR/TA results will effectively be ignored in the final action determination.

Next, the boundary subset TR/TA results are combined to determine an overall status of the object. If the object is TR against any viewport or enabled UC plane, the object is considered TR. Note that, by definition, being TR against a VPXY boundary implies that the vertices will be TR agains the corresponding GB boundary, so computing TR_GB is unnecessary.

The treatment of the UCF outcodes is conditional on the UserClipFlags MustClip Enable state. If DISABLED, an object that is not TR against the UCFs is considered TA against them. Put another way, objects will only be culled (not clipped) with respect to the UCFs. If ENABLED, the UCF outcodes are treated like the other outcodes, in that they are used to determine TR, TA or MC status, and an object can be passed to a CLIP thread simply based on it straddling a UCF.

Finally, the object is considered MC if it is neither TR or TA.

The following logic is used to compute the final TR, TA, and MC status.

```
//
// ClipDetermination
//
// Compute objANDCode and objORCode
//
switch (object type) {
   case POINT:
      {
         objANDCode[...] = v0.outCode[...]
         objORCode[...]  = v0.outCode[...]
      } break
   case LINE:
      {
         objANDCode[...] = v0.outCode[...] & v1.outCode[...]
         objORCode[...]  = v0.outCode[...] | v1.outCode[...]
      } break
   case TRIANGLE:
      {
         objANDCode[...] = v0.outCode[...] & v1.outCode[...] & v2.outCode[...]
         objORCode[...]  = v0.outCode[...] | v1.outCode[...] | v2.outCode[...]
      } break
}
//
// Determine TR/TA against interesting boundary subsets
//
TR_VPXY = (objANDCode[VP_L] | objANDCode[VP_R] | objANDCode[VP_T] | objANDCode[VP_B])
TR_GB   = (objANDCode[GB_L] | objANDCode[GB_R] | objANDCode[GB_T] | objANDCode[GB_B])
TA_GB   = !(objORCode[GB_L] | objORCode[GB_R]  | objORCode[GB_T]  | objORCode[GB_B])
TA_VPZ  = !(objORCode[VP_N] | objORCode[VP_Z])
TR_VPZ  = (objANDCode[VP_N] | objANDCode[VP_Z])
TA_UC   = !(objORCode[UC0]  | objORCode[UC1]   | ... | objORCode[UC7])
TR_UC   = (objANDCode[UC0]  | objANDCode[UC1]  | ... | objANDCode[UC7])
BAD     = objORCode[BAD]
TA_NEGW = !objORCode[NEGW]
TR_NEGW = objANDCode[NEGW]

//
```

```
// Trivial Reject
//
// An object is considered TR if all vertices are TR against any common boundary
// Note that this allows the case of the VPXY being outside the GB
//
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC || TR_NEGW
#else
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC

//
// Trivial Accept
//
// For an object to be TA, it must be TA against the VPZ and GB, not TR,
// and considered TA against the UC planes and NEGW
// If the UCMC mode is disabled, an object is considered TA against the UC
// as long as it isn't TR against the UC.
// If the UCMC mode is enabled, then the object really has to be TA against the UC
// to be considered TA
// In this way, enabling the UCMC mode will force clipping if the object is neither
// TA or TR against the UC
//
TA   = !TR && TA_GB && TA_VPZ && TA_NEGW
UCMC = CLIP_STATE.UserClipFlagsMustClipEnable
TA   = TA && ( (UCMC && TA_UC) || (!UCMC && !TR_UC) )

//
// MustClip
// This is simply defined as not TA or TR
// Note that exactly one of TA, TR and MC will be set
//
MC = !(TA || TR)
```

ClipMode State

The ClipMode state determines what action the CLIP unit takes given the results of ClipDetermination. The possible actions are:

- **PASSTHRU:** Pass the object directly down the pipeline. A CLIP thread is not spawned.
- **DISCARD:** Remove the object from the pipeline and dereference object vertices as required (that is, dereferencing will not occur if the vertices are shared with other objects).
- **SPAWN:** Pass the object to a CLIP thread. In the process of initiating the thread, the object vertices may be dereferenced.

The following logic is used to determine what to do with the object (PASSTHRU or DISCARD or SPAWN).

```
//
// Use the ClipMode to determine the action to take
//
switch (CLIP_STATE.ClipMode) {
   case NORMAL:
      {
         PASSTHRU = TA && !BAD
         DISCARD = TR || BAD
         SPAWN   = MC && !BAD
      }
   case CLIP_ALL:
      {
         PASSTHRU = 0
```

```
            DISCARD  = 0
            SPAWN    = 1
         }
      case CLIP_NOT_REJECT:
         {
            PASSTHRU = 0
            DISCARD  = TR   || BAD
            SPAWN    = !(TR || BAD)
         }
      case REJECT_ALL:
         {
            PASSTHRU = 0
            DISCARD  = 1
            SPAWN    = 0
         }
      case ACCEPT_ALL:
         {
            PASSTHRU = !BAD
            DISCARD  = BAD
            SPAWN    = 0
         }
   } endswitch
```

NORMAL ClipMode

In NORMAL mode, objects will be discarded if TR or BAD, passed through if TA, and passed to a CLIP thread if MC. Those mode is typically used when the CLIP kernel is only used to perform 3D Clipping (the expected usage model).

CLIP_ALL ClipMode

In CLIP_ALL mode, all objects (regardless of classification) will be passed to CLIP threads. Note that this includes BAD objects. This mode can be used to perform arbitrary processing in the CLIP thread, or as a backup if for some reason the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are not sufficient for controlling 3D Clipping.

CLIP_NON_REJECT ClipMode

This mode is similar to CLIP_ALL mode, but TR and BAD objects are discarded an all other (TA, MC) objects are passed to CLIP threads. Usage of this mode assumes that the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are sufficient at least in respect to determining trivial reject.

REJECT_ALL ClipMode

In REJECT_ALL mode, all objects (regardless of classification) are discarded. This mode effectively clips out all objects.

ACCEPT_ALL ClipMode

In ACCEPT_ALL mode, all non-BAD objects are passed directly down the pipeline. This mode partially disables the CLIP stage. BAD objects will still be discarded, and incomplete primitives (generated by a GS thread) will be discarded.

Primitive topologies with adjacency are also handled, in that the adjacent-only vertices are dereferenced and only non-adjacent objects are passed down the pipeline. This condition can arise when primitive topologies with adjacency are generated but the GS stage is disabled. If this condition is allowed, the CLIP stage must not be completely disabled – as this would allow adjacent vertices to pass through the CLIP stage and lead to unpredictable results as the rest of the pipeline does not comprehend adjacency.

Object Pass-Through

Depending on ClipMode, objects may be passed directly down the pipeline. The PrimTopologyType associated with the output objects may differ from the input PrimTopologyType, as shown in the table below.

Programming Note: The CLIP unit does *not* tolerate primitives with adjacency that have "dangling vertices". This should not be an issue under normal conditions, as the VF unit does not generate these sorts of primitives and the GS thread is restricted (though by specification only) to not output these sorts of primitives.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
POINTLIST	POINTLIST	
POINTLIST_BF	POINTLIST_BF	
LINELIST	LINELIST	
LINELIST_ADJ	LINELIST	Adjacent vertices removed.
LINESTRIP	LINESTRIP	
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT	LINESTRIP	Adjacent vertices removed. LINESTRIP_ADJ_CONT is added for BDW+. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINESTRIP_BF	LINESTRIP_BF	
LINESTRIP_CONT	LINESTRIP_CONT	
LINESTRIP_CONT_BF	LINESTRIP_CONT_BF	
LINELOOP	N/A	Not supported after GS.
TRILIST	TRILIST	
RECTLIST	RECTLIST	
TRILIST_ADJ	TRILIST	Adjacent vertices removed.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
TRISTRIP	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping subsection.
TRISTRIP_REV	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects. See Tristrip Clipping subsection.
TRISTRIP_ADJ	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects. Adjacent vertices removed. See Tristrip Clipping subsection.
TRIFAN	TRIFAN	
TRIFAN_NOSTIPPLE	TRIFAN_NOSTIPPLE	
POLYGON, POLYGON_CONT	POLYGON	POLYGON_CONT is added for BDW+. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
QUADLIST	N/A	Not supported after GS.
QUADSTRIP	N/A	Not supported after GS.

Primitive Output

(This section refers to output from the CLIP unit to the pipeline, not output from the CLIP thread)

The CLIP unit will output primitives (either passed-through or generated by a CLIP thread) in the proper order. This includes the buffering of a concurrent CLIP thread's output until the preceding CLIP thread terminates. Note that the requirement to buffer subsequent CLIP thread output until the preceding CLIP thread terminates has ramifications on determining the number of VUEs allocated to the CLIP unit and the number of concurrent CLIP threads allowed.

Other Functionality

Statistics Gathering

The CLIP unit includes logic to assist in the gathering of certain pipeline statistics . The statistics take the form of MI counter registers (see Memory Interface Registers), where the CLIP unit provides signals causing those counters to increment.

Software is responsible for controlling (enabling) these counters in order to provide the required statistics at the DDI level. For example, software might need to disable statistics gathering before submitting non-API-visible objects (e.g., RECTLISTs) for processing.

The CLIP unit must be ENABLED (via the CLIP Enable bit of PIPELINED_STATE_POINTERS) for it to affect the statistics counters. This might lead to a pathological case where the CLIP unit needs to be ENABLED simply to provide statistics gathering. If no clipping functionality is desired, Clip Mode can be set to ACCEPT_ALL to effectively inhibit clipping while leaving the CLIP stage ENABLED.

The statistic the CLIP unit affects (if enabled) is CL_INVOCATION_COUNT, incremented for every object received from the GS stage.

CL_INVOCATION_COUNT

If the **Statistics Enable** bit (CLIP_STATE) is set, the CLIP unit increments the CL_INVOCATION_COUNT register for every complete object received from the GS stage.

To maintain a count of application-generated objects, software must clear the CLIP unit's **Statistic Enable** whenever driver-generated objects are rendered.

3D Pipeline - Strips and Fans (SF) Stage

The Strips and Fan (SF) stage of the 3D pipeline is responsible for performing "setup" operations required to rasterize 3D objects.

This functionality is handled completely in hardware, and the SF unit no longer has the ability to spawn threads.

Inputs from CLIP

The following table describes the per-vertex inputs passed to the SF unit from the previous (CLIP) stage of the pipeline.

SF's Vertex Pipeline Inputs

Variable	Type	Description
primType	enum	Type of primitive topology the vertex belongs to. *Primitive Assembly* for a list of primitive types supported by the SF unit. See *3D Pipeline* for descriptions of these topologies. Notes: The CLIP unit will convert any primitive with adjacency (3DPRIMxxx_ADJ) it receives from the pipeline into the corresponding primitive without adjacency (3DPRIMxxx). QUADLIST, QUADSTRIP, LINELOOP primitives are not supported by the SF unit. Software must use a GS thread to convert these to some other (supported) primitive type.

Variable	Type	Description
		If an object is clipped by the hardware clipper, the CLunit would force this field to "3DPRIM_POLYGON". SFunit would process this incoming object just as it would any other "3DPRIM_POLYGON". SFunit selects vertex 0 as the provoking vertex.
primStart,primEnd	boolean	Indicate vertex's position within the primitive topology
vInX[]	float	Vertex X position (screen space or NDC space)
vInY[]	float	Vertex Y position (screen space or NDC space)
vInZ[]	float	Vertex Z position (screen space or NDC space)
vInInvW[]	float	Reciprocal of Vertex homogeneous (clip space) W
hVUE[]	URB address	Points to the vertex's data stored in the URB (one VUE handle per vertex)
renderTargetArrayIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in subsequent processing.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in the Viewport Transform and Scissor subfunctions, otherwise the value is ignored. Note that for primitive topologies with vertices shared between objects, this means a shared vertex may be subject to multiple Viewport Transformation operations if the viewPortIndex varies within the topology.
pointSize	uint	If this vertex is within a POINTLIST[_BF] primitive topology, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

Attribute Setup/Interpolation Process

The following sections describe the Attribute Setup/Interpolation Process.

Attribute Setup/Interpolation Process

Hardware computes all needed parameters, as there is no setup thread.

Outputs to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Coefficients for Z, 1/W, perspective and non-perspective b1 and b2 per vertex, and attribute vertex deltas a0, a1, and a2 per attribute.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.).
- Edge or line interpolation information (e.g., edge equation coefficients, etc.).
- Information on where the WM is to start rasterization of the object.
- Object orientation (front/back-facing).
- Last Pixel indication (for line drawing).

Primitive Assembly

The first subfunction within the SF unit is *Primitive Assembly*. Here 3D primitive vertex information is buffered and, when a sufficient number of vertices are received, converted into basic 3D objects which are then passed to the Viewport Transformation subfunction.

The number of vertices passed with each primitive is constrained by the primitive type. *Primitive Assembly*. Passing any other number of vertices results in UNDEFINED behavior. Note that this restriction only applies to primitive output by GS threads (which is under control of the GS kernel). See the Vertex Fetch chapter for details on how the VF unit automatically removes incomplete objects resulting from processing a 3DPRIMITIVE command.

SF-Supported Primitive Types & Vertex Count Restrictions

primType	VertexCount Restriction
3DPRIM_TRILIST	nonzero multiple of 3
3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE	>=3
3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	>=3
3DPRIM_LINELIST	nonzero multiple of 2
3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT	>=2

primType	VertexCount Restriction
3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	
3DPRIM_RECTLIST	nonzero multiple of 3
3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	nonzero

Primitive Assembly for a list of the 3D object types.

3D Object Types

objectType	generated by primType	Vertices/Object
3DOBJ_POINT	3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	1
3DOBJ_LINE	3DPRIM_LINELIST 3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	2
3DOBJ_TRIANGLE	3DPRIM_TRILIST 3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE 3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	3
3DOBJ_RECTANGLE	3DPRIM_RECTLIST	3 (expanded to 4 in RectangleCompletion)

Primitive Assembly for the outputs of Primitive Decomposition.

Primitive Decomposition Outputs

Variable	Type	Description
objectType	enum	Type of object. *Primitive Assembly*
nV	uint	The number of object vertices passed to Object Setup. *Primitive Assembly*
v[0..nV-1]*	various	Data arrays associated with object vertices. Data in the array consists of X, Y, Z, invW and a pointer to the other vertex attributes. These additional attributes are not used by directly by the 3D fixed functions but are made available to the SF thread. The number of valid vertices depends on the object type. *Primitive Assembly*

Variable	Type	Description
invertOrientation	enum	Indicates whether the orientation (CW or CCW winding order) of the vertices of a triangle object should be inverted. Ignored for non-triangle objects.
backFacing	enum	Valid only for points and line objects, indicates a back facing object. This is used later for culling.
provokingVtx	uint	Specifies the index (into the *v[]* arrays) of the vertex considered the "provoking" vertex (for flat shading). The selection of the provoking vertex is programmable via SF_STATE (**xxx Provoking Vertex Select** state variables.)
polyStippleEnable	boolean	TRUE if Polygon Stippling is enabled. FALSE for TRIFAN_NOSTIPPLE. Ignored for non-triangle objects.
continueStipple	boolean	Only applies to line objects. TRUE if Line Stippling should be continued (i.e., not reset) from where the previous line left off. If FALSE, Line Stippling is reset for each line object.
renderTargetIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. This value is simply passed in SF thread payloads and not used within the SF unit.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object.
pointSize	unit	For point objects, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

The following table defines, for each primitive topology type, which vertex's VPIndex/RTAIndex applies to the objects within the topology.

VPIndex/RTAIndex Selection

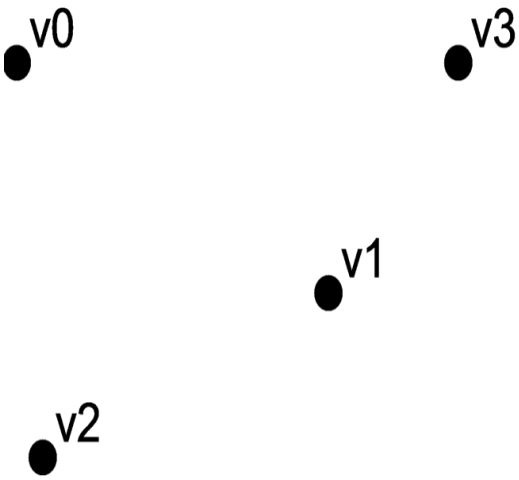
PrimTopologyType	Viewport Index Usage
POINTLIST POINTLIST_BF	Each vertex supplies the VPIndex for the corresponding point object
LINELIST	The leading vertex of each line supplies the VPIndex for the corresponding line object. V0.VPIndex→ Line(V0,V1) V2.VPIndex→ Line(V2,V3) ...
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF	The leading vertex of each line segment supplies the VPIndex for the corresponding line object. V0.VPIndex→ Line(V0,V1) V1.VPIndex→ Line(V1,V2) ...

PrimTopologyType	Viewport Index Usage
	NOTE: If the VPIndex changes within the topology, shared vertices will be processed (mapped) multiple times.
TRILIST RECTLIST	The leading vertex of each triangle/rect supplies the VPIndex for the corresponding triangle/rect objects. V0.VPIndex→ Tri(V0,V1,V2) V3.VPIndex→ Tri(V3,V4,V5) ... NOTE: For Autostrips multiple viewport index is not supported. APIs defines the multiple viewport index to be output from Geometry shader that only generates Tristrips. If any other shader outputing multiple viewport indices for other topologies either autostrip needs to be disable or clipper guardband test needs to be disabled.
TRISTRIP TRISTRIP_REVERSE	The leading vertex of each triangle supplies the VPIndex for the corresponding triangle object. V0.VPIndex→ Tri(V0,V1,V2) V1.VPIndex→ Tri(V1,V2,V3) ... NOTE: If the VPIndex changes within the primitive, shared vertices will be processed (mapped) multiple times.
TRIFAN TRIFAN_NOSTIPPLE POLYGON	The first vertex (V0) supplies the VPIndex for all triangle objects.

Point List Decomposition

The 3DPRIM_POINTLIST and 3DPRIM_POINTLIST_BACKFACING primitives specify a list of independent points.

3DPRIM_POINTLIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_POINT objects that are then passed individually and in order to the Object Setup subfunction. The *provokingVertex* of each object is, by definition, v[0].

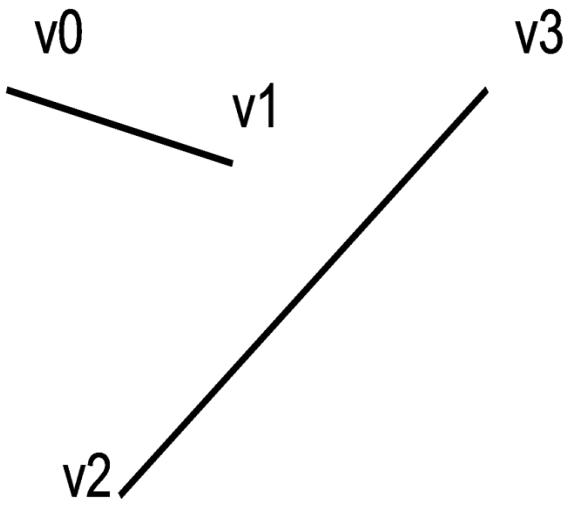
Points have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing points. Primitives of type 3DPRIM_POINTLIST_BACKFACING are decomposed exactly the same way as 3DPRIM_POINTLIST primitives, but the *backFacing* variable is set for resulting point objects being passed on to object setup.

```
PointListDecomposition()
{
   objectType = 3DOBJ_POINT
   nV = 1
   provokingVtx = 0
   if (primType == 3DPRIM_POINTLIST)
      {
         backFacing = FALSE
      }
   else // primType == 3DPRIM_POINTLIST_BACKFACING
      {
         backFacing = TRUE
      }

   for each (vertex in [0..vertexCount-1])
      {
         v[0] ← vIn[i] // copy all arrays
                       // (for example, v[]X, v[]Y, and so on)
         ObjectSetup()
      }
}
```

Line List Decomposition

The 3DPRIM_LINELIST primitive specifies a list of independent lines.

3DPRIM_LINELIST Primitive



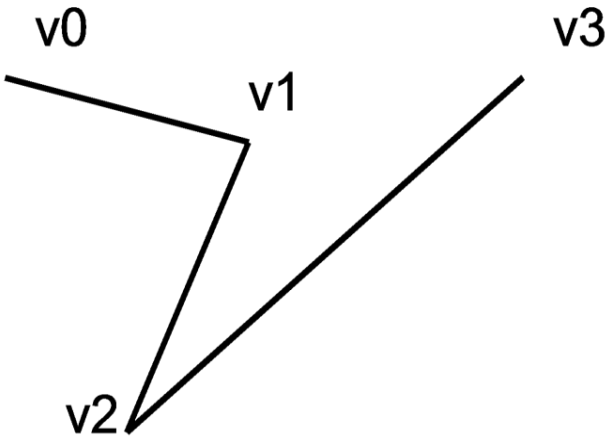
The decomposition process divides the list into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0, v1; v2, v3; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```
LineListDecomposition()
   {
      objectType = 3DOBJ_LINE
      nV = 2
      provokingVtx = Line List/Strip Provoking Vertex Select       continueStipple = FALSE
      for each (vertex I in [0..vertexCount-2] by 2)
         {
             v[0] arrays ← vIn[i] arrays
             v[1] arrays ← vIn[i+1] arrays
             ObjectSetup()
         }
   }
```

Line Strip Decomposition

The 3DPRIM_LINESTRIP, 3DPRIM_LINESTRIP_CONT, 3DPRIM_LINESTRIP_BF, and 3DPRIM_LINESTRIP_CONT_BF primitives specify a list of connected lines.

3DPRIM_LINESTRIP_xxx Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0,v1; v1,v2; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

Lines have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing lines. Primitives of type 3DPRIM_LINESTRIP[_CONT]_BF are decomposed exactly the same way as 3DPRIM_LINESTRIP[_CONT] primitives, but the *backFacing* variable is set for the resulting line objects being passed on to object setup. Likewise 3DPRIM_LINESTRIP_CONT[_BF] primitives are decomposed identically to basic line strips, but the *continueStipple* variable is set to true so that the line stipple pattern will pick up from where it left off with the last line primitive, rather than being reset.

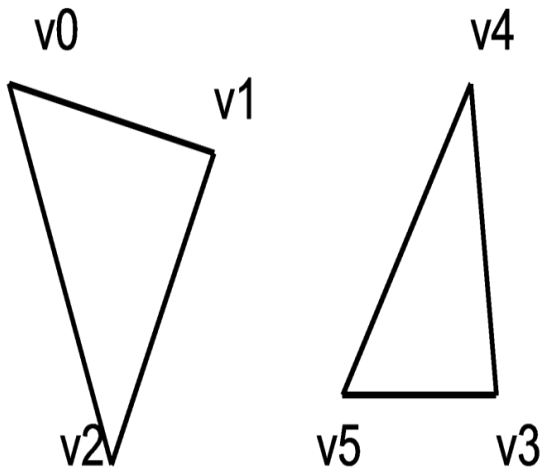
```
LineStripDecomposition()
   {
      objectType = 3DOBJ_LINE
      nV = 2
      provokingVtx = Line List/Strip Provoking Vertex Select
      if (primType == 3DPRIM_LINESTRIP)
         {
             backFacing = FALSE
             continueStipple = FALSE
         } else if (primType == 3DPRIM_LINESTRIP_BF)
            {
               backFacing = TRUE
               continueStipple = FALSE
            } else if (primType == 3DPRIM_LINESTRIP_CONT)
               {
                  backFacing = FALSE
                  continueStipple = TRUE
               } else if (primType == 3DPRIM_LINESTRIP_CONT_BF)
                  {
                     backFacing = TRUE
                     continueStipple = TRUE
                  }
      for each (vertex I in [0..vertexCount-1])
         {
             v[0] arrays ← vIn[i] arrays
```

```
            v[1] arrays ← vIn[i+1] arrays
            ObjectSetup()
            continueStipple = TRUE
        }
    }
```

Triangle List Decomposition

The 3DPRIM_TRILIST primitive specifies a list of independent triangles.

3DPRIM_TRILIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v3,v4,v5; and so on. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

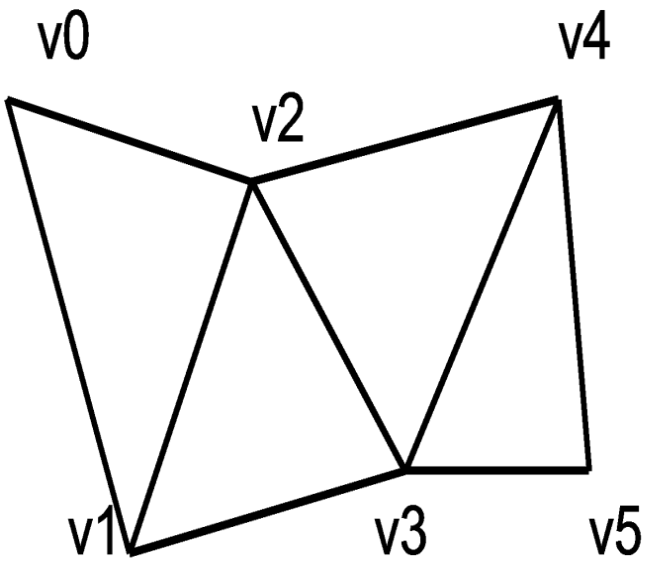
```
TriangleListDecomposition() {
  objectType = 3DOBJ_TRIANGLE
      nV = 3
  invertOrientation = FALSE
  provokingVtx = Triangle List/Strip Provoking Vertex Select
  polyStippleEnable = TRUE
  for each (vertex I in [0..vertexCount-3] by 3)
      {

          v[0] arrays ⟵ vIn[i] arrays


          v[1] arrays ⟵ vIn[i+1] arrays


          v[2] arrays ⟵ vIn[i+2] arrays

          ObjectSetup()
```

```
        }
}
```

Triangle Strip Decomposition

The 3DPRIM_TRISTRIP and 3DPRIM_TRISTRIP_REVERSE primitives specify a series of triangles arranged in a strip, as illustrated below.

3DPRIM_TRISTRIP[_REVERSE] Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v1,v2,v3; v2,v3,v4; and so on. Note that the *winding order* of the vertices alternates between CW (clockwise), CCW (counter-clockwise), CW, etc. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

The 3D pipeline uses the winding order of the vertices to distinguish between front-facing and back-facing triangles (*Triangle Orientation (Face) Culling* below). Therefore, the 3D pipeline must account for the alternation of winding order in strip triangles. The *invertOrientation* variable is generated and used for this purpose.

To accommodate the situation where the driver is forced to break an input strip primitive into multiple tristrip primitive commands (for example, due to ring or batch buffer size restrictions), two tristrip primitive types are supported. 3DPRIM_TRISTRIP is used for the initial section of a strip, and wherever a continuation of a strip starts with a triangle with a CW winding order. 3DPRIM_TRISTRIP_REVERSE is used for a continuation of a strip that starts with a triangle with a CCW winding order.

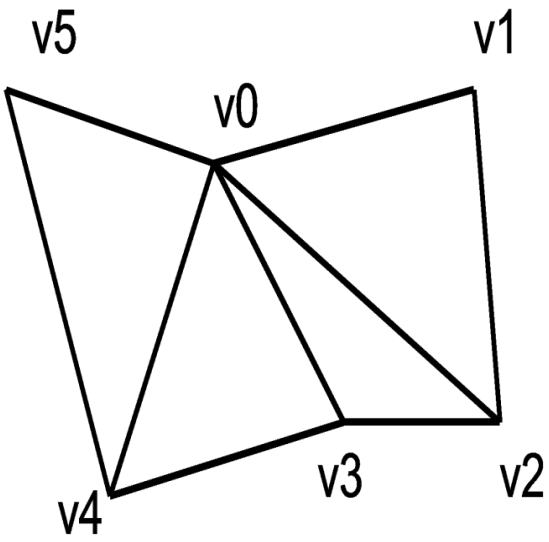
```
TriangleStripDecomposition()
   {
```

```
        objectType = 3DOBJ_TRIANGLE
        nV = 3
        provokingVtx = Triangle List/Strip Provoking Vertex Select
        if (primType == 3DPRIM_TRISTRIP)
           invertOrientation = FALSE
           else // primType == 3DPRIM_TRISTRIP_REVERSE
              invertOrientation = TRUE
        polyStippleEnable = TRUE
        for each (vertex I in [0..vertexCount-3])
          {
              v[0] arrays ← vIn[i] arrays
              v[1] arrays ← vIn[i+1] arrays
              v[2] arrays ← vIn[i+2] arrays
              ObjectSetup()
              invertOrientation = ! invertOrientation
          }
    }
```

Triangle Fan Decomposition

The 3DPRIM_TRIFAN and 3DPRIM_TRIFAN_NOSTIPPLE primitives specify a series of triangles arranged in a fan, as illustrated below.

3DPRIM_TRIFAN Primitive



The decomposition process divides the fan into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v0,v2,v3; v0,v3,v4; and so on. As there is no alternation in the vertex winding order, the *invertOrientation* variable is output as FALSE unconditionally. The *provokingVertex* of each object is taken from the **Triangle Fan Provoking Vertex** state variable, as programmed via SF_STATE.

Primitives of type 3DPRIM_TRIFAN_NOSTIPPLE are decomposed exactly the same way, except the *polyStippleEnable* variable is FALSE for the resulting objects being passed on to object setup. This will inhibit polygon stipple for these triangle objects.

```
TriangleFanDecomposition()
   {
       objectType = 3DOBJ_TRIANGLE
       nV = 3
       invertOrientation = FALSE
       provokingVtx = Triangle Fan Provoking Vertex Select
       if (primType == 3DPRIM_TRIFAN)
           polyStippleEnable = TRUE
           else // primType == 3DPRIM_TRIFAN_NOSTIPPLE
              polyStippleEnable = FALSE

       v[0] arrays ← vIn[0] arrays
         // the 1st vertex is common

       for each (vertex I in [1..vertexCount-2])
          {
              v[1] arrays ← vIn[i] arrays
              v[2] arrays ← vIn[i+1] arrays
              ObjectSetup()
          }
   }
```

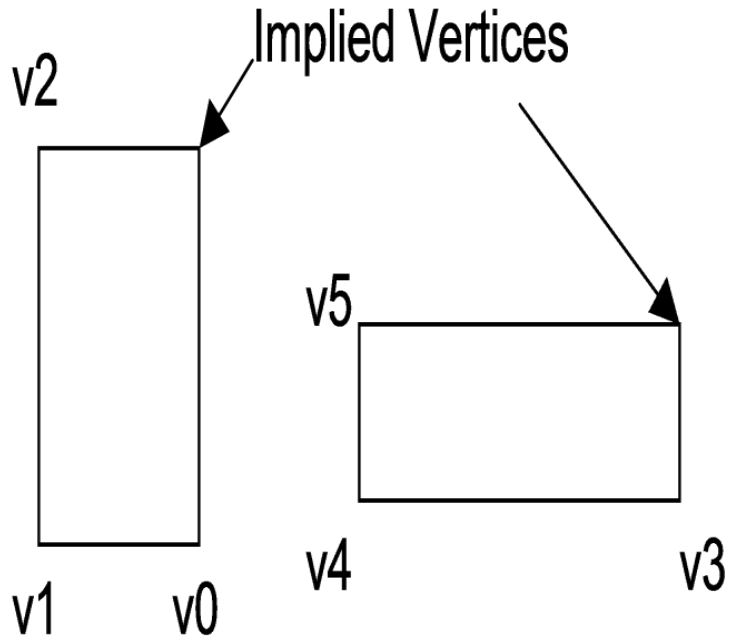
Polygon Decomposition

The 3DPRIM_POLYGON primitive is identical to the 3DPRIM_TRIFAN primitive with the exception that the *provokingVtx* is overridden with 0. This support has been added specifically for OpenGL support, avoiding the need for the driver to change the provoking vertex selection when switching between trifan and polygon primitives.

Rectangle List Decomposition

The 3DPRIM_RECTLIST primitive command specifies a list of independent, axis-aligned rectangles. Only the lower right, lower left, and upper left vertices (in that order) are included in the command – the upper right vertex is derived from the other vertices (in Object Setup).

3DPRIM_RECTLIST Primitive



The decomposition of the 3DPRIM_RECTLIST primitive is identical to the 3DPRIM_TRILIST decomposition, with the exception of the *objectType* variable.

```
RectangleListDecomposition()
   {
       objectType = 3DOBJ_RECTANGLE
       nV = 3
       invertOrientation = FALSE
       provokingVtx = 0
       for each (vertex I in [0..vertexCount-3] by 3)
          {
              v[0] arrays ←  vIn[i] arrays
              v[1] arrays ←  vIn[i+1] arrays
              v[2] arrays ←  vIn[i+2] arrays
              ObjectSetup()
          }
   }
```

Object Setup

The Object Setup subfunction of the SF stage takes the post-viewport-transform data associated with each vertex of a basic object and computes various parameters required for scan conversion. This includes generation of implied vertices, translations and adjustments on vertex positions, and culling (removal) of certain classes of objects. The final object information is passed to the Windower/Masker (WM) stage where the object is rasterized into pixels.

Invalid Position Culling (Pre/Post-Transform)

At input the the SF stage, any objects containing a floating-point NaN value for Position X, Y, Z, or RHW will be unconditionally discarded. Note that this occurs on an object (not primitive) basis.

If Viewport Transformation is enabled, any objects containing a floating-point NaN value for post-transform Position X, Y or Z will be unconditionally discarded.

Viewport Transformation

If the **Viewport Transform Enable** bit of SF_STATE is ENABLED, a viewport transformation is applied to each vertex of the object.

The VPIndex associated with the leading vertex of the object is used to obtain the **Viewport Matrix Element** data from the corresponding element of the SF_VIEWPORT structure in memory. For each object vertex, the following scale and translate transformation is applied to the position coordinates:

$x' = \mathbf{m00} * x + \mathbf{m30}$

$y' = \mathbf{m11} * y + \mathbf{m31}$

$z' = \mathbf{m22} * z + \mathbf{m32}$

Software is responsible for computing the matrix elements from the viewport information provided to it from the API.

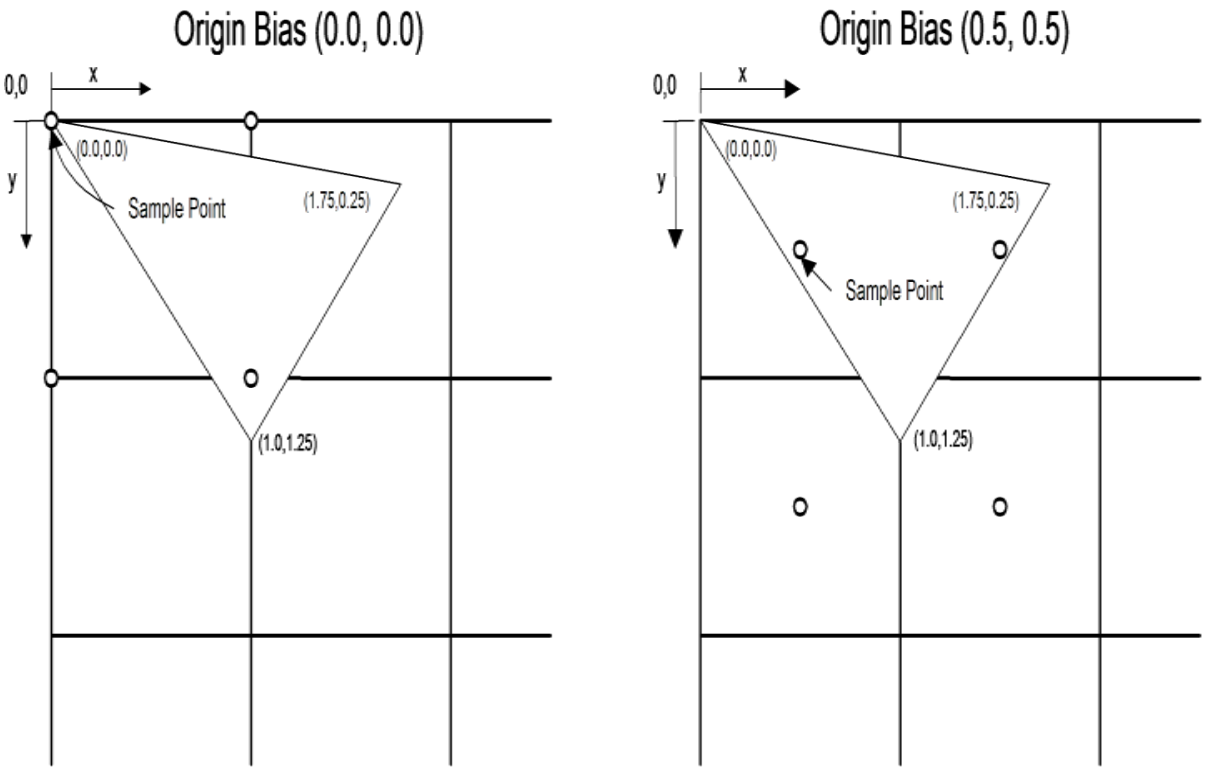
Destination Origin Bias

The positioning of the pixel sampling grid is programmable and is controlled by the **Destination Origin Horizontal/Vertical Bias** state variables (set via SF_STATE). If these bias values are both 0, pixels are sampled on an integer grid. Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0,0) falls within the primitive.

If the bias values are both 0.5, pixels are sampled on a “half” integer grid (i.e., X.5, Y.5). Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0.5,0.5) falls within the primitive. This positioning of the sample grid corresponds with the OpenGL rasterization rules, where “fragment centers” lay on a half-integer grid. It also corresponds with the Intel740 rasterizer (though that device did not employ “top left” rules).

Note that subsequent descriptions of rasterization rules for the various objects will be with reference to the pixel sampling grid.

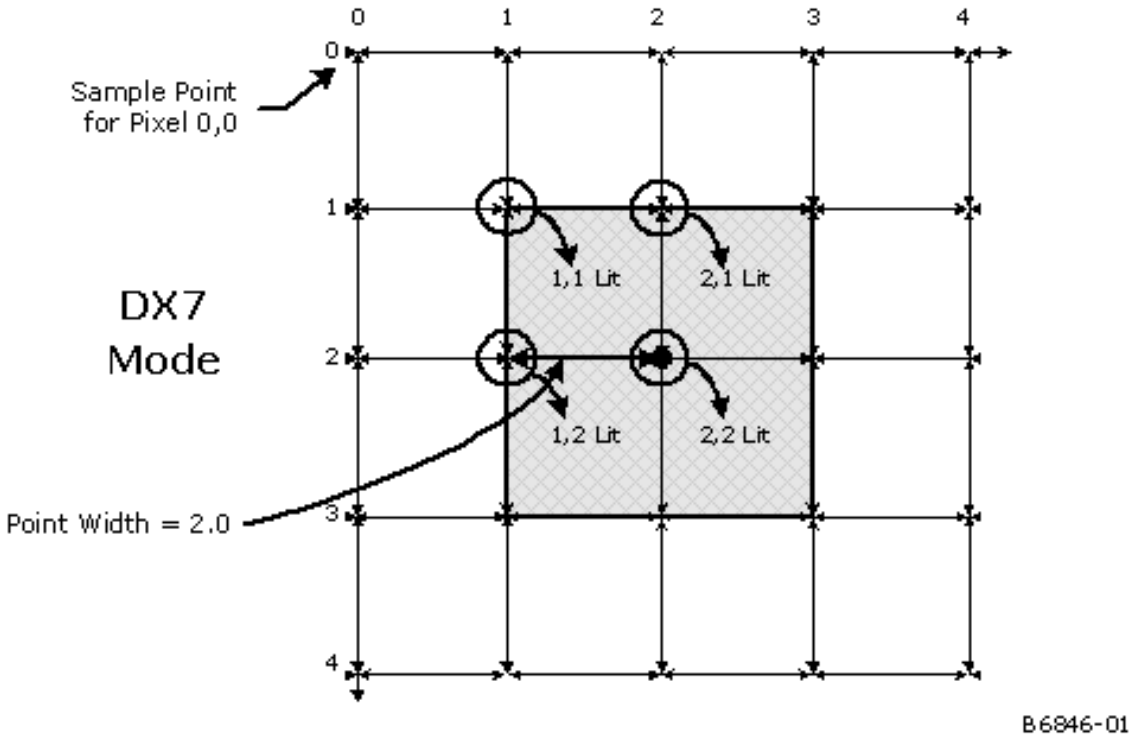
Destination Origin Bias



Point Rasterization Rule Adjustment

POINT objects are rasterized as square RECTANGLEs, with one exception: The **Point Rasterization Rule** state variable (in SF_STATE) controls the rendering of point object edges that fall directly on pixel sample points, as the treatment of these edge pixels varies between APIs.

RASTRULE_UPPER_LEFT

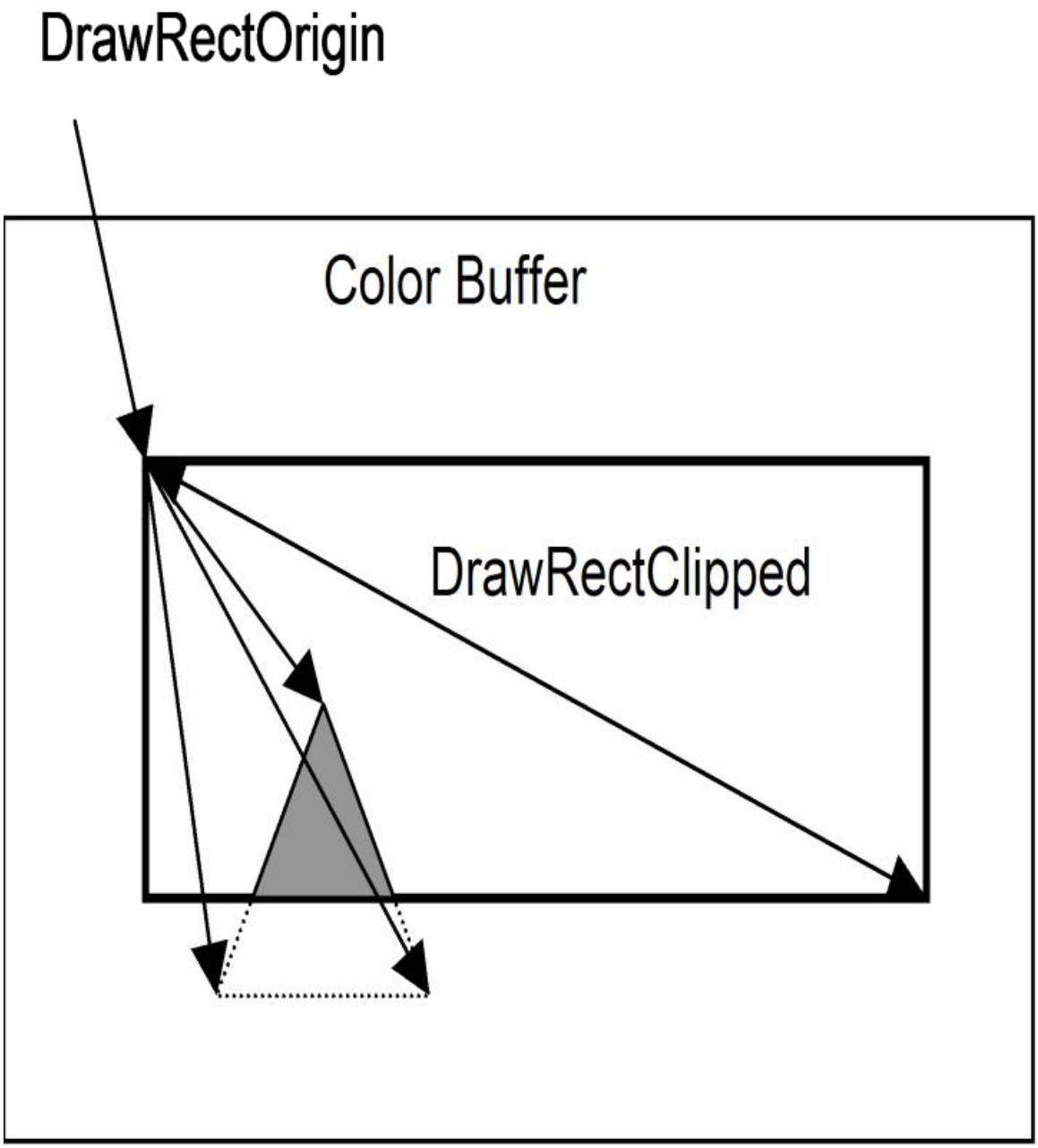


Drawing Rectangle Offset Application

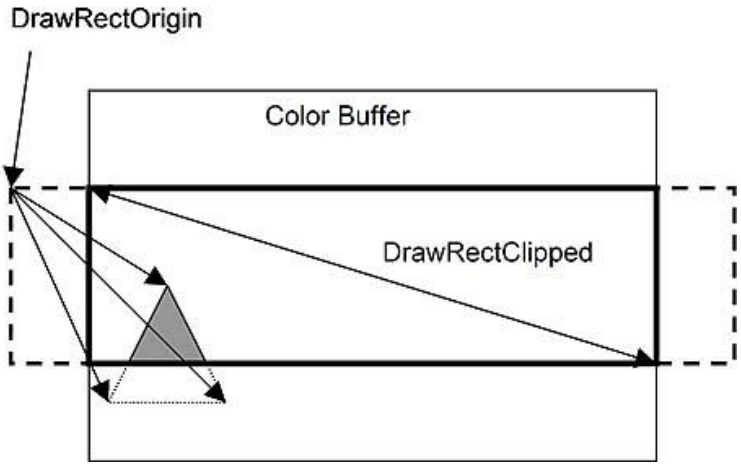
The Drawing Rectangle Offset subfunction offsets the object's vertex X,Y positions by the pixel-exact, unclipped drawing rectangle origin (as programmed via the **Drawing Rectangle Origin X,Y** values in the 3DSTATE_DRAWING_RECTANGLE command). The Drawing Rectangle Offset subfunction (at least with respect to Color Buffer access) is unconditional, and therefore to (effectively) turn off the offset function the origin would need to be set to (0,0). A non-zero offset is typically specified when window-relative or viewport-relative screen coordinates are input to the device. Here the drawing rectangle origin would be loaded with the absolute screen coordinates of the window's or viewport's upper-left corner.

Clipping of objects which extend outside of the Drawing Rectangle occurs later in the pipeline. Note that this clipping is based on the "clipped" draw rectangle (as programmed via the **Clipped Drawing Rectangle** values in the 3DSTATE_DRAWING_RECTANGLE command), which must be clamped by software to the rendertarget boundaries. The unclipped drawing rectangle origin, however, can extend outside the screen limits in order to support windows whose origins are moved off-screen. This is illustrated in the following diagrams.

Onscreen Draw Rectangle



Partially-offscreen Draw Rectangle



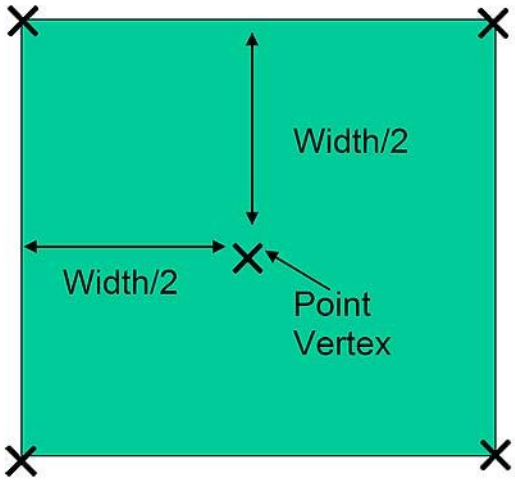
3DSTATE_DRAWING_RECTANGLE

Point Width Application

This stage of the pipeline applies only to 3DOBJ_POINT objects. Here the point object is converted from a single vertex to four vertices located at the corners of a square centered at the point's X,Y position. The width and height of the square are specified by a *point width* parameter. The **Point Width Source** value in SF_STATE determines the source of the point width parameter: the point width is either taken from the **Point Width** value programmed in SF_STATE or the PointWidth specified with the vertex (as read back from the vertex VUE earlier in the pipeline).

The corner vertices are computed by adding and subtracting one half of the point width. *Point Width Application*.

Point Width Application

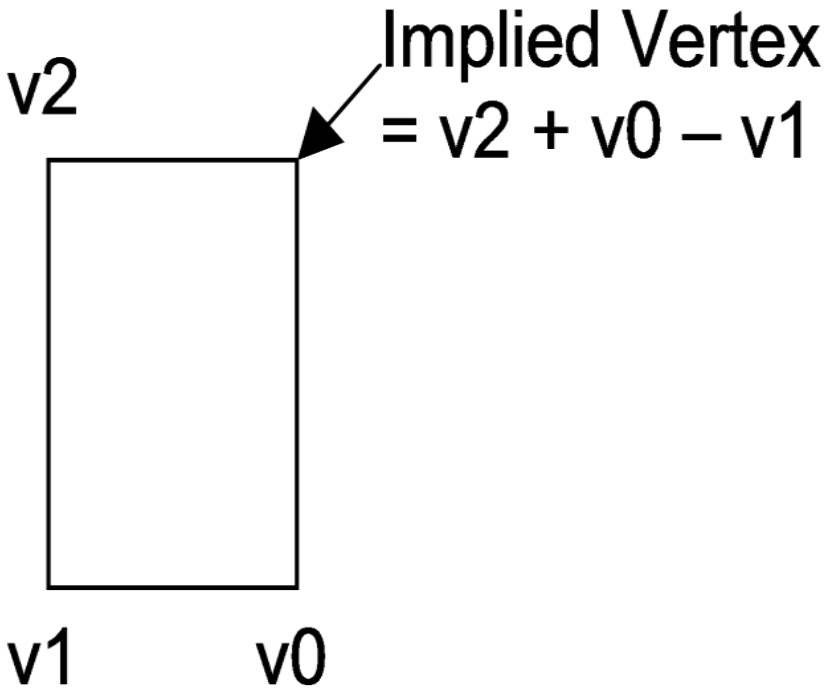


Z and W vertex attributes are copied from the single point center vertex to each of the four corner vertices.

Rectangle Completion

This stage of the pipeline applies only to 3DOBJ_RECTANGLE objects. Here the X,Y coordinates of the 4th (upper right) vertex of the rectangle object is computed from the first 3 vertices as shown in the following diagram. The other vertex attributes assigned to the implied vertex (v[3]) are UNDEFINED as they are not used. The Object Setup subfunction will use the values at only the first 3 vertices to compute attribute interpolants used across the entire rectangle.

Rectangle Completion



Vertex XY Clamping and Quantization

At this stage of the pipeline, vertex X and Y positions are in continuous screen (pixel) coordinates. These positions are quantized to subpixel precision by rounding the incoming values to the nearest subpixel (using round-to-nearest-or-even rules matching the DirectX reference device). The device supports rasterization with either 4 or 8 fractional (subpixel) position bits, as specified by the **Vertex SubPixel Precision Select** bit of SF_STATE.

The vertex X and Y screenspace coordinates are also *clamped* to the fixed-point "guardband" range supported by the rasterization hardware, as listed in the following table:

Per-Device Guardband Extents

Supported X,Y ScreenSpace "Guardband" Extent	Maximum Post-Clamp Delta (X or Y)
[-32K,32K-1]	N/A

Note that this clamping occurs after the Drawing Rectangle Origin has been applied and objects have been expanded (i.e., points have been expanded to squares, etc.). In almost all circumstances, if an object's vertices are actually modified by this clamping (i.e., had X or Y coordinates outside of the guardband extent the rendered object will not match the intended result. Therefore software should take steps to ensure that this does not happen – e.g., by clipping objects such that they do not exceed these limits after the Drawing Rectangle is applied.

In addition, in order to be correctly rendered, objects must have a screenspace bounding box not exceeding 8K in the X or Y direction. This additional restriction must also be comprehended by software, i.e., enforced by use of clipping.

Degenerate Object Culling

At this stage of the pipeline, "degenerate" objects are discarded. This operation is automatic and cannot be disabled. (The object rasterization rules would by definition cause these objects to be "invisible" – this culling operation is mentioned here to reinforce that the device implementation optimizes these degeneracies as early as possible).

Degenerate Object Culling for definitions of degenerate objects.

Degenerate Objects

objType	Degenerate Object Definition
3DOBJ_POINT	Two or more corner vertices are coincident (i.e., the radius quantized to zero)
3DOBJ_LINE	The endpoints are coincident
3DOBJ_TRIANGLE	All three vertices are collinear or any two vertices are coincident and SOLID fill mode applies to the triangle
3DOBJ_RECTANGLE	Two or more corner vertices are coincident

Triangle Orientation (Face) Culling

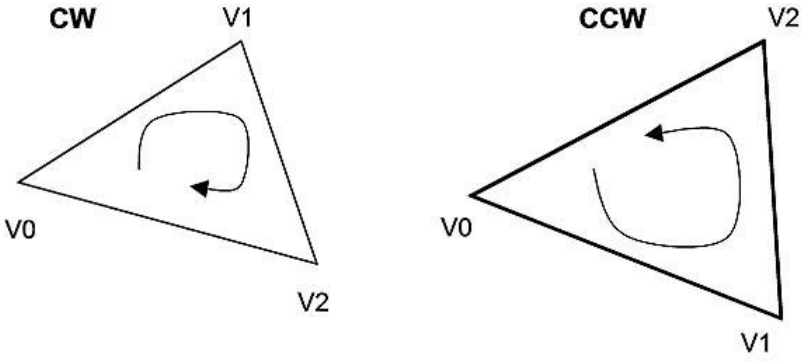
At this stage of the pipeline, 3DOBJ_TRIANGLE objects can be optionally discarded based on the "face orientation" of the object. This culling operation does not apply to the other object types.

This operation is typically called "back face culling", though front facing objects (or all 3DOBJ_TRIANGLE objects) can be selected to be discarded as well. Face culling is typically used to eliminate triangles facing away from the viewer, thus reducing rendering time.

The "winding order" of a triangle is defined by the the triangle vertex's 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal proceeds in either a clockwise (CW) or counter-clockwise (CCW) direction. The "winding order" of a triangle is defined by the the triangle vertex's 2D

(X,Y) screen space position when traversed from v0 to v1 to v2. That traversal will proceed in either a clockwise (CW) or counter-clockwise (CCW) direction. A degenerate triangle is considered "backfacing", regardless of the FrontWinding state.

Triangle Winding Order



The **Front Winding** state variable in SF_STATE controls whether CW or CCW triangles are considered as having a "front-facing" orientation (at which point non-front-facing triangles are considered "back-facing"). The internal variable *invertOrientation* associated with the triangle object is then used to determine whether the orientation of a that triangle should be inverted. Recall that this variable is set in the Primitive Decomposition stage to account for the alternating orientations of triangles in strip primitives resulting form the ordering of the vertices used to process them.

The **Cull Mode** state variable in SF_STATE specifies how triangles are discarded according to their resultant orientation. See Degenerate Objects.

Cull Mode

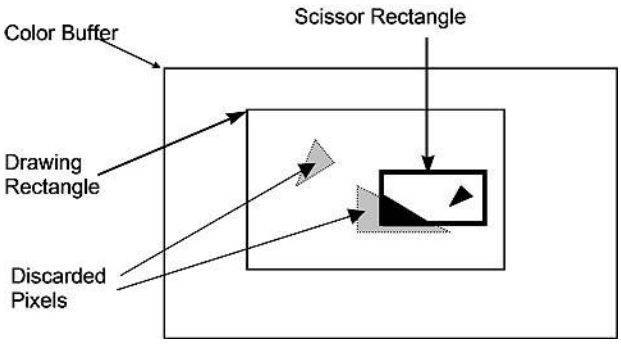
CullMode	Definition
CULLMODE_NONE	The face culling operation is disabled.
CULLMODE_FRONT	Triangles with "front facing" orientation are discarded.
CULLMODE_BACK	Triangles with "back facing" orientation are discarded.
CULLMODE_BOTH	All triangles are discarded.

Scissor Rectangle Clipping

A *scissor* operation can be used to restrict the extent of rendered pixels to a screen-space aligned rectangle. If the scissor operation is enabled, portions of objects falling outside of the intersection of the scissor rectangle and the clipped draw rectangle are clipped (pixels discarded).

The scissor operation is enabled by the **Scissor Rectangle Enable** state variable in SF_STATE. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding SF_VIEWPORT structure. Up to 16 structures are supported. The **Scissor Rectangle X,Y Min,Max** fields of the SF_VIEWPORT structure defines a scissor rectangle as a rectangle in integer pixel coordinates relative to the (unclipped) origin of the Drawing Rectangle. The scissor rectangle is defined relative to the Drawing Rectangle to better support the OpenGL API. (OpenGL specifies the "Scissor Box" in window-

relative coordinates). This allows instruction buffers with embedded Scissor Rectangle definitions to remain valid even after the destination window (drawing rectangle) moves.



Specifying either scissor rectangle xmin > xmax or ymin > ymax will cause all polygons to be discarded for a given viewport (effectively a null scissor rectangle).

Viewport Extents Test

Viewport extents test can be used to restrict the extent of rendered pixels to the viewport extents. If this operation is enabled, portion of the objects falling outside of the intersection of the scissor rectangle (if enabled) and the clipped draw rectangle and viewport extents are clipped (pixels discarded). This operation similar to the scissor test except both have different enables and the viewport extents can be programmed to the fractional float values.

This operation is enabled by the **View Transform Enable** state variable in **SF_STATE**. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding **SF_CLIP_VIEWPORT** structure. Up to 16 structures are supported. The X/Y **Min/Max ViewPort** fields of the **SF_CLIP_VIEWPORT** structure defines viewport extents as a rectangle in float screen pixel coordinates relative to the (unclipped) origin of the Drawing Rectangle. Please note that these co-ordinates can be fractional values and hardware will do appropriate rounding and convert it to integer pixel co-ordinates (floor rouding used). This **View Transform Enable** state also controls the viewport transform so appropriate the viewport transform coefficients need to be populated in the SF_CLIP_VIEPWORT structure along with the viewport extents.

Final clip rectangle used to define the rendering area will now depend on three rectangles namely drawing rectangle, Scissor rectangle, Viewport Extents. If both **Scissor Rectangle Enable** and **View transform enable** are set then intersection of all rectangles (Viewport extents, Scissor rectangle, Draw rectangle) becomes final clip rectangle, while If only **Scissor Rectangle Enable** is enabled then the intersection of (Scissor rectangle, Draw rectangle) becomes final clip rectangle. If only **View transform enable** is enabled then intersection of (Viewport extents, Draw rectangle) become the final clip rectangle, while If none is enabled then (Draw rectangle) is the final clip rectangle.

Specifying either viewport extents xmin > xmax or ymin > ymax will cause all polygons to be discarded for a given viewport (effectively a null viewport).

Line Rasterization

The device supports three styles of line rendering: *zero-width* (*cosmetic*) lines, *non-antialiased* lines, and *antialiased* lines. Non-antialiased lines are rendered as a polygon having a specified width as measured parallel to the major axis of the line. Antialiased lines are rendered as a rectangle having a specified width measured perpendicular to the line connecting the vertices.

The functions required to render lines are split between the SF and WM units. The SF unit is responsible for computing the overall geometry of the object to be rendered, including the pixel-exact bounding box, edge equations, etc., and therefore is provided with the screen-geometry-related state variables. The WM unit performs the actual scan conversion, determining the exact pixels included/excluded and coverage values for anti-aliased lines.

Zero-Width (Cosmetic) Line Rasterization

Note: The specification of zero-width line rasterization would be more correctly included in the WM Unit chapter, though is being included here to keep it with the rasterization details of the other line types.

When the **Line Width** is set to zero, the device will use special rules to rasterize zero-width ("cosmetic") lines. The **Anti-Aliasing Enable** state variable is ignored when **Line Width** is zero.

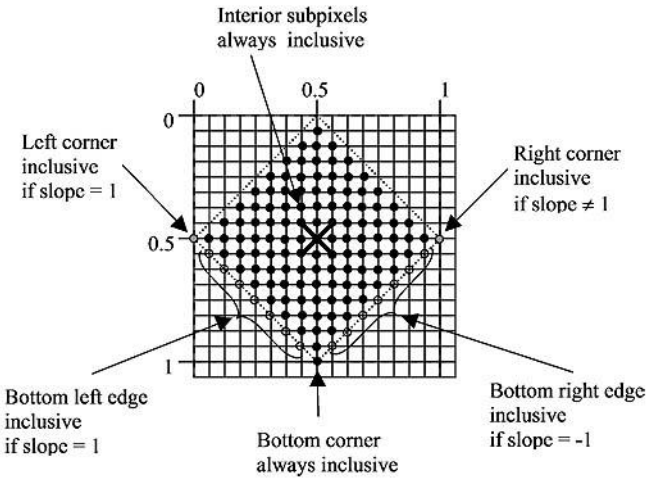
When the **LineWidth** is set to zero, the device will use special rules to rasterize "cosmetic" lines. The rasterization rules also comply with the OpenGL conformance requirements (for 1-pixel wide non-smooth lines). Refer to the appropriate API specifications for details on these requirements.

The GIQ rules basically intersect the directed, ideal line connecting two endpoints with an array of diamond-shaped areas surrounding pixel sample points. Wherever the line exits a diamond (including passing through a diamond), the corresponding pixel is lit. Special rules are used to define the subpixel locations that are considered interior to the diamonds, as a function of the slope of the line. When a line ends in a diamond (and therefore does not exit that diamond), the corresponding pixel is not drawn. When a line starts in a diamond and exits that diamond, the corresponding pixel is drawn.

GIQ (Diamond) Sampling Rules – Legacy Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is ENABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered "interior" to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line slope is not exactly one, in which case the left corner subpixel is inclusive.** Including the right corner subpixel ensures that lines with slopes in the range (1, +infinity] or [-infinity, -1) touch a diamond even when they cross exactly between pixel diamonds. Including the left corner on slope=1 lines is required for proper handling of slope=1 lines (see (3) below) – where if the right corner was inclusive, a slope=1 line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
3. **The subpixels along the bottom left edge are inclusive only if the line slope = 1.** This is to correctly handle the case where a slope=1 line falls enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this "passing through" the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One slope=1 line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would case the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as "inside" for slope=1 lines, we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.
4. **The subpixels along the bottom right edge are inclusive only if the line slope = -1.** Similar case as (3), except slope=-1 lines require the bottom right edge to be considered inclusive.

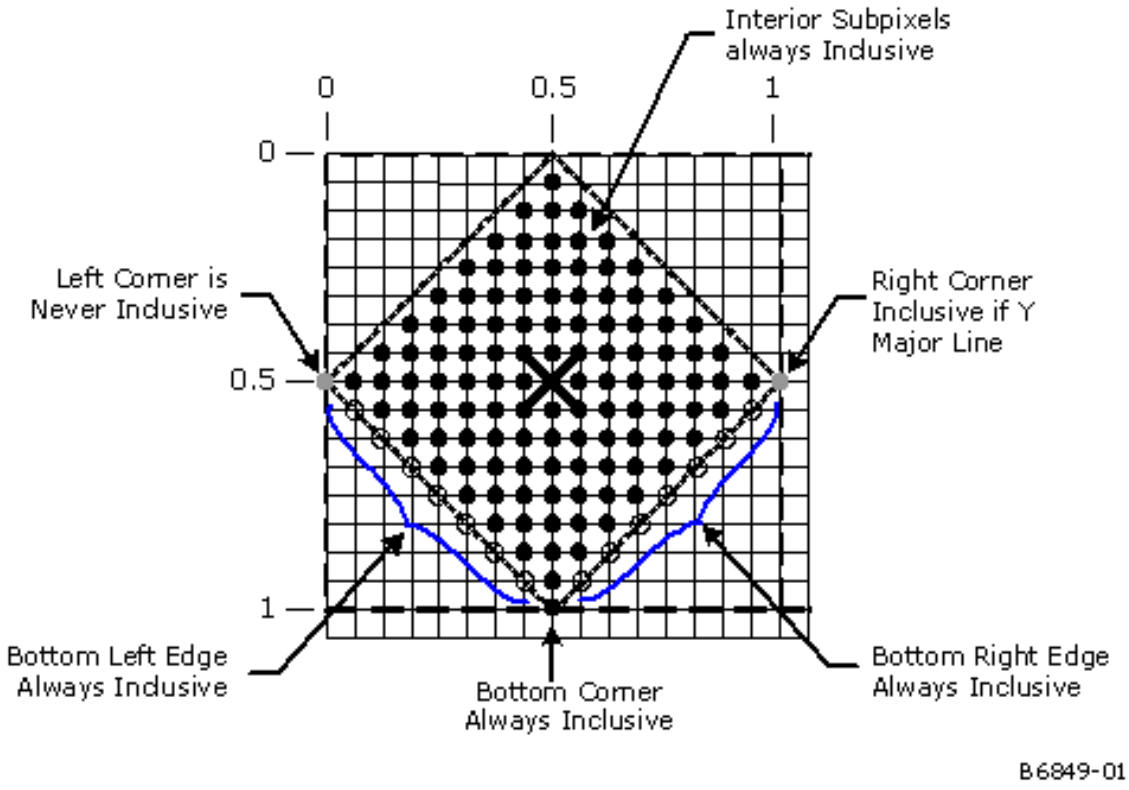
The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (slopePosOne, slopeNegOne).

```
      delta_x                          = point.x - sample.x
      delta_y                          = point.y - sample.y
      distance                   = abs(delta_x) + abs(delta_y)
      interior                   = (distance < 0.5)
      bottom_corner              = (delta_x == 0.0) && (delta_y == 0.5)
      left_corner                = (delta_x == -0.5) && (delta_y == 0.0)
      right_corner               = (delta_x == 0.5) && (delta_y == 0.0)
      bottom_left_edge           = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)
      bottom_right_edge          = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

      inside = interior || bottom_corner || (slopePosOne ? left_corner : right_corner) ||
(slopePosOne && left_edge) || (slopeNegOne && right_edge)
```

GIQ (Diamond) Sampling Rules – DX10 Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is DISABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered "interior" to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line is not X Major (X Major is defined as -1 <= slope <= 1).** Including the right corner subpixel ensures that lines with slopes in the range (>1, +infinity] or [-infinity, <-1) touch a diamond even when they cross exactly between pixel diamonds.
3. **The left corner subpixel is never inclusive.** For Y Major lines, having the right corner subpixel as always inclusive requires that the left corner subpixel should never be inclusive, since a line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
4. **The subpixels along the bottom left edge are always inclusive.** This is to correctly handle the case where a line enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this "passing through" the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would cause the pixel to be lit twice – breaking the rule that connected line segments

should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as "inside", the pixel is only lit when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.

5. **The subpixels along the bottom right edge are always inclusive.** Same as case as (4), except slope=-1 lines require the bottom right edge to be considered inclusive.

The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (XMajor).

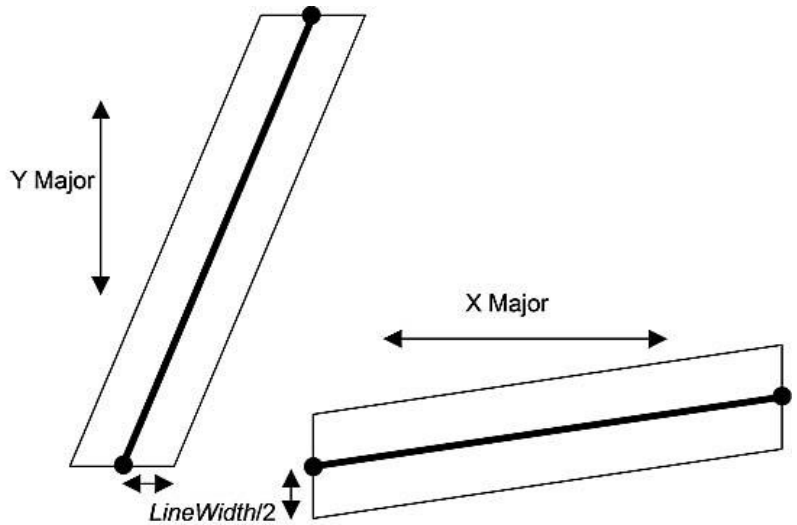
```
delta_x                         = point.x - sample.x
delta_y                         = point.y - sample.y
distance                        = abs(delta_x) + abs(delta_y)
interior                        = (distance < 0.5)
bottom_corner                         = (delta_x == 0.0)  && (delta_y == 0.5)
left_corner                     = (delta_x == -0.5) && (delta_y == 0.0)
right_corner                    = (delta_x == 0.5)  && (delta_y == 0.0)
bottom_left_edge                = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)
bottom_right_edge               = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior || bottom_corner || (!XMajor && right_corner) || ( bottom_left_edge)
|| ( bottom_right_edge)
```

Non-Antialiased Wide Line Rasterization

Non-anti-aliased, non-zero-width lines are rendered as parallelograms that are centered on, and aligned to, the line joining the endpoint vertices. Pixels sampled interior to the parallelogram are rendered; pixels sampled exactly on the parallelogram edges are rendered according to the polygon "top left" rules.

The parallelogram is formed by first determining the major axis of the line (diagonal lines are considered x-major). The corners of the parallelogram are computed by translating the line endpoints by +/-(**Line Width** / 2) in the direction of the minor axis, as shown in the following diagram.

Non-Antialiased Line Rasterization

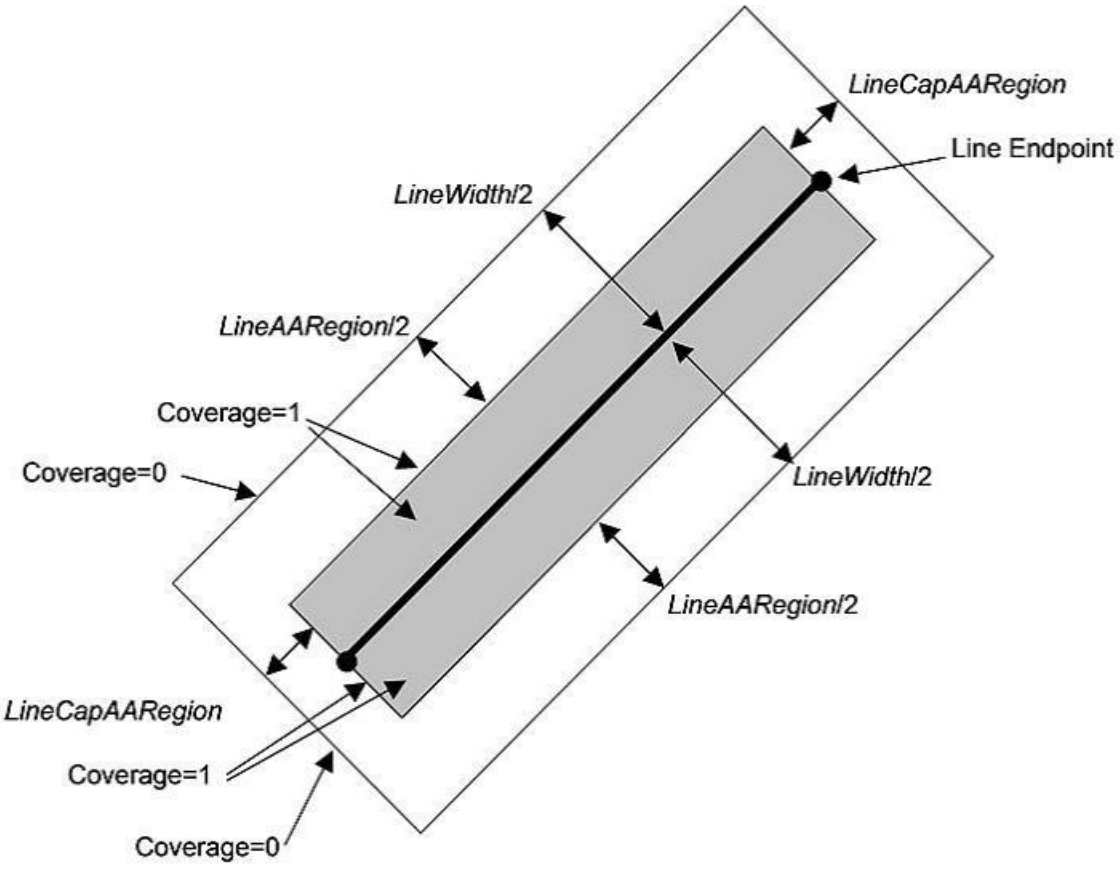


Anti-Aliased Line Rasterization

Anti-aliased lines are rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices. For each pixel in the rectangle, a fractional coverage value (referred to as Antialias Alpha) is computed – this coverage value is normally used to attenuate the pixel's alpha in the pixel shader thread. The resultant alpha value is therefore available for use in those downstream pixel pipeline stages to generate the desired effect (e.g., use the attenuated alpha value to modulate the pixel's color, and add the result to the destination color, etc.). Note that software is required to explicitly program the pixel shader and pixel pipeline to obtain the desired anti-aliasing effect – the device simply makes the coverage-attenuated pixel alpha values available for use in the pixel shader.

The dimensions of the rendered rectangle, and the parameters controlling the coverage value computation, are programmed via the **Line Width**, **Line AA Region**, and **Line Cap AA Region** state variables, as shown below. The edges parallel to the line are located at the distance (*LineWidth*/2) from the line (measured in screen pixel units perpendicular to the line). The end-cap edges are perpendicular to the line and located at the distance (*LineCapAARegion*) from the endpoints.

Anti-aliased Line Rasterization



Along the parallel edges, the coverage values ramp from the value 0 at the very edges of the rectangle to the value 1 at the perpendicular distance (*LineAARegion*/2) from a given edge (in the direction of the line). A pixel's coverage value is computed with respect to the closest edge. In the cases where

(*LineAARegion*/2) < (*LineWidth*/2), this results in a region of fractional coverage values near the edges of the rectangle, and a region of "fully-covered" coverage values (i.e., the value 1) at the interior of the line. When (*LineAARegion*/2) == (*LineWidth*/2), only pixel sample points falling exactly on the line can generate fully-covered coverage values. If (*LineAARegion*/2) > (*LineWidth*/2), no pixels can be fully-covered (it is expected that this case is not typically desired).

Along the end cap edges, the coverage values ramp from the value 1 at the line endpoint to the value 0 at the cap edge – itself at a perpendicular distance (*LineCapAARegion*) from the endpoint. Note that, unlike the line-parallel edges, there is only a single parameter (*LineCapAARegion*) controlling the extension of the line at the end caps and the associated coverage ramp.

The regions near the corners of the rectangle have coverage values influenced by distances from both the line-parallel and end cap edges – here the two coverage values are multiplied together to provide a composite coverage value.

The computed coverage value for each pixel is passed through the Windower Thread Dispatch payload. The Pixel Shader kernel should be passed (unmodified) by the shader to the Render Cache as part of it's output message.

SF Pipeline State Summary

3DSTATE_RASTER

3DSTATE_RASTER

Signal	SF_INT::Multisample Rasterization Mode
Description	This field determines whether multisample rasterization is enabled and how pixel sample points are defined.
Formula	See Table: WM_INT::Multisample Rasterization Mode in 3D Pipeline Windower > Windower Pipelined State > 3DSTATE_WM > 3DSTATE_WM

Signal	SF_INT::Global Depth Offset Enable Solid
Description	This field determines when Global Depth bias gets enabled.
Formula	= (3DSTATE_RASTER:: Global Depth Offset Enable Solid? ((3DSTATE_RASTER::Global Depth Offset Constant != IEEE_FP_ZERO) || (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disable

Signal	SF_INT:: Global Depth Offset Enable Wireframe
Description	This field determines when Global Depth bias gets enabled.
Formula	= (3DSTATE_RASTER:: Global Depth Offset Enable Wireframe? ((3DSTATE_RASTER::Global Depth Offset Constant != IEEE_FP_ZERO) || (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disable

Signal	SF_INT:: Global Depth Offset Enable Point
Description	This field determines when Global Depth bias gets enabled.
Formula	= (3DSTATE_RASTER:: Global Depth Offset Enable Point? ((3DSTATE_RASTER::Global De IEEE_FP_ZERO) || (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disab

3DSTATE_SF

The state used by the SF stage is defined by this inline state packet.

3DSTATE_SF

The SF Unit also receives 3DSTATE_RASTER. It also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_WM, 3DSTATE_WM_HZ_OP, 3DSTATE_DETPH_BUFFER, and 3DSTATE_MULTISAMPLE fields.

Signal	SF_INT::Number of Multisamples
Description	Set the number of multisamples.
Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP::Number of Multisamples : 3DSTATE_MULTISAMPLE::Number of Multisamples

Signal	SF_INT::Pixel Position Offset Enable
Description	Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions.

Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP:: Pixel Position Offset Enable: 3DSTATE_MULTISAMPLE:: Pixel Position Offset Enable

Signal	SF_INT::Pixel Position Offset
Description	Causes the device to offset pixel positions by 0.5 both in horizontal and vertical directions. It is to be noted this is done to adjust the pixel co-ordinate system to DX9 like, so any screen space rectangles (eg: HiZ Clear, Resolve etc) generated internally by driver in this mode needs to be aware of this offset adjustment and send the rectangles according to alignment restriction taking this offset adjustment into consideration.
Formula	= (SF_INT::Number of Multisamples >1) && (3DSTATE_MULTISAMPLE:: Pixel Location == PIXLOC_UL_CORNER) && SF_INT::Pixel Position Offset Enable

Signal	SF_INT: Global Depth Offset Enable Solid
Description	Set the number of multisamples
Formula	= (3DSTATE_RASTER:: Global Depth Offset Enable Solid ? ((3DSTATE_RASTER::Global Depth Offset Constant != IEEE_FP_ZERO) ‖ (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disable

Signal	SF_INT: Global Depth Offset Enable Wireframe
Description	Set the number of multisamples
Fromula	= (3DSTATE_RASTER:: Global Depth Offset Enable Wireframe ? ((3DSTATE_RASTER::Global Depth Offset Constant != IEEE_FP_ZERO) ‖ (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disable

Signal	SF_INT: Global Depth Offset Enable Point
Description	Set the number of multisamples.
Formula	= (3DSTATE_RASTER:: Global Depth Offset Enable Point ? ((3DSTATE_RASTER::Global Depth Offset Constant != IEEE_FP_ZERO) || (3DSTATE_RASTER::Global Depth Offset Scale != IEEE_FP_ZERO)): Disable

Signal	SF_INT::FrontFace Fill Mode
Description	This state controls how front-facing triangle and rectangle objects are rendered.
Formula	= 3DSTATE_INT::WM_HZ_OP ? SOLID : 3DSTATE_RASTER:: FrontFace Fill Mode

Signal	SF_INT::BackFace Fill Mode
Description	This state controls how Back-facing triangle and rectangle objects are rendered.
Formula	= 3DSTATE_INT::WM_HZ_OP ? SOLID : 3DSTATE_RASTER:: BackFace Fill Mode

Signal	SF_INT::FrontWinding
Description	Determines whether a triangle object is considered “front facing” if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.
Formula	= 3DSTATE_INT::WM_HZ_OP ? FRONTWINDING_CW : 3DSTATE_RASTER::FrontWinding

Signal	SF_INT::Cull Mode
Description	Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points or rectangles.

Formula	= SF_INT::WM_HZ_OP ? CULLMODE_BACK : 3DSTATE_RASTER:: Cull Mode

Signal	SF_INT::Scissor Rectangle Enable
Description	This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.
Formula	= SF_INT::WM_HZ_OP ? 3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable : 3DSTATE_RASTER::Scissor Rectangle Enable

Signal	SF_INT::Anti-aliasing Enable
Description	This field enables “alpha-based” line antialiasing.
Formula	= = SF_INT::WM_HZ_OP ? 3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable : 3DSTATE_RASTER::Anti-aliasing Enable

Signal	SF_INT::Global Depth Offset Constant
Description	Specifies the constant term in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Constant

Signal	SF_INT::Global Depth Offset Scale
Description	Specifies the constant term in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Scale

Signal	SF_INT::Global Depth Offset Clamp
Description	Specifies the clamp term used in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Clamp

Signal	SF_INT::Line Stipple Enable
Description	Specifies the clamp term used in the Global Depth Offset function.
Formula	= 3DSTATE_WM::Line Stipple Enable

Signal	SF_INT::RT Independent Rasterization Enable
Description	Enables RT Independent Rasterization.

Formula	= 3DSTATE_INT::WM_HZ_OP ? Disable : 3DSTATE_RASTER::ForcedSampleCount != NUMRASTSAMPLES_0

Signal	SF_INT::WM_HZ_OP
Description	Enables WM_HZ_OP.
Formula	= (3DSTATE_WM_HZ_OP::DepthBufferClear || 3DSTATE_WM_HZ_OP::DepthBufferResolve || 3DSTATE_WM_HZ_OP::Hierarchical Depth Buffer Resolve Enable || 3DSTATE_WM_HZ_OP::StencilBufferClear) ? Enable : Disable

Signal	SF_INT:: View Transform Enable
Description	Enables View Transform
Formula	= SF_INT::WM_HZ_OP ? Disable : 3DSTATE_SF::View Transform Enable

Signal	SF_INT::Render Target Array index
Description	Render Target Array index being render to
Formula	= 3DSTATE_INT::WM_HZ_OP ? 0 : Render Target Array index pipelined from clipper

Signal	SF_INT::Depth Buffer Surface Format
Description	Depth format being used
Formula	= 3DSTATE_INT:: Depth Buffer Surface Format

Signal	SF_INT::Viewport index
Description	Viewport being used.
Formula	= SF_INT::WM_HZ_OP ? 0 :

	Viewport index pipelined from clipper

SF_CLIP_VIEWPORT

The viewport-specific state used by both the SF and CL units (SF_CLIP_VIEWPORT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 16 DWords apart. The location of the first element of the array, as specified by both **Pointer to SF_VIEWPORT** and **Pointer to CLIP_VIEWPORT**, is aligned to a 64-byte boundary.

SCISSOR_RECT

Attribute Interpolation Setup

With the attribute interpolation setup function being implemented in hardware for , a number of state fields in 3DSTATE_SF are utilized to control interpolation setup.

Number of SF Output Attributes sets the number of attributes that will be output from the SF stage, not including position. This can be used to specify up to 32, and may differ from the number of input attributes. The number of input attributes is derived from the **Vertex URB Entry Read Length** field. Note that this field is also used to specify whether swizzling is to be performed on Attributes 0-15 or Attributes 16-32. See the state field definition for details.

Attribute Swizzling

The first or last set of 16 attributes can be swizzled according to certain state fields. **Attribute Swizzle Enable** enables the swizzling for all 16 of these attributes, and each of the attributes has a 2-bit **Swizzle Select** field that controls swizzling with the following settings:

- INPUTATTR – This attribute is sourced from AttrInputReg[SourceAttribute].
- INPUTATTR_FACING – This attribute is sourced from AttrInputReg[SourceAttribute] if the object is front-facing, otherwise it is sourced from AttrInputReg[SourceAttribute+1].
- INPUTATTR_W – This attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination).
- INPUTATTR_FACING – If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination). If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. WYZW.

Each of the first or last set of 16 attributes also has a 5-bit **Source Attribute** field which specify, per output attribute (not component), which input attribute sources the output attribute when INPUTATTR is selected for **Swizzle Select**. A **Source Attribute** value of 0 corresponds to the 128-bit attribute immediately following the vertex 4D position. If INPUTATTR_FACING is selected, this specifies the first of two consecutive (front,back) input attributes, where the SourceAttribute value can be an odd or even number (just not 31, as that would place the back-face input attribute past the end of the input max complement of input attributes).

Constant overriding is also available for the first or last set of 16 attributes. Each attribute has a **Constant Source** field which specifies the constant values per swizzled attribute, with the following settings available:

- XYZW = 0000
- XYZW = 0001
- XYZW = 1111

Each channel of each attribute has a **Component Override** field to control whether the corresponding channel is overridden with the constant value defined in **Constant Source**.

Interpolation Modes

All 32 attributes have a **Constant Interpolation Enable** state field bit to specify whether all components of the post-swizzled attribute are to be interpolated as constant values (not varying over the pixels of the object). If set, the attribute at the provoking vertex is copied to a0, and a1 and a2 are set to zero – this results in a constant interpolation of the provoking vertex value. If clear, the attribute is linearly interpolated. Attributes 0-15 are further subjected to Wrap Shortest processing on a per-component basis, via the **Attribute WrapShortest Enables** state bitfields. WrapShortest processing modifies the a1 and/or a2 values depending on attribute deltas. All

The table below indicates the output values of a0, a1, and a2 depending on interpolation mode settings.

	a0	a1	a2
Constant	A0	0.0	0.0
Linear	A0	A1-A0	A2-A0
Wrap Shortest	A0	(A1-A0)+1 (A1-A0) <= -0.5 (A1-A0)-1 (A1-A0) >= 0.5 (A1-A0) otherwise	(A2-A0)+1 (A2-A0) <= -0.5 (A2-A0)-1 (A2-A0) >= 0.5 (A2-A0) otherwise

Point Sprites

Normally all vertex attributes (including texture coordinates) other than position are simply replicated from the incoming point center vertex to the generated point object (corner) vertices. However, both DX9 and OGL support “sprite points”, where some/all texture coordinates are replaced with full-scale 2D texture coordinates.

A 32-bit **PointSprite TextureCoordinate Enable** bit mask controls whether the corresponding vertex attribute is to be replaced by a sprite point texture coordinate. The global (not per-attribute) **Point Sprite TextureCoordinate Origin** field controls how the point object vertex (top/bottom, left/right) texture coordinates are generated:

UPPERLEFT	Left	Right
Top	(0,0,0,1)	(1,0,0,1)
Bottom	(0,1,0,1)	(1,1,0,1)

LOWERLEFT	Left	Right
Top	(0,1,0,1)	(1,1,0,1)
Bottom	(0,0,0,1)	(1,0,0,1)

The state used by “setup backend” is defined by the following inline state packet.

3DSTATE_SBE

The state used by “setup backend” is defined by the following inline state packet.

3DSTATE_SBE_SWIZ

SBE Unit also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_VS, 3DSTATE_DS, and 3DSTATE_GS fields.

Signal	SBE_INT::Vertex URB Entry Read Length
Description	Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments.
Formula	= (3DSTATE_SBE::Force Vertex URB Entry Read Length == Force) ? 3DSTATE_SBE::Vertex URB Entry Read Length : 3DSTATE_GS::GS_Enable ? 3DSTATE_GS::Vertex URB Entry Output Length : 3DSTATE_DS::DS_Enable ? 3DSTATE_DS::Vertex URB Entry Output Length : 3DSTATE_VS::Vertex URB Entry Output Length

Signal	SBE_INT::Vertex URB Entry Read Offset
Description	Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB
Formula	= (3DSTATE_SBE::Force Vertex URB entry Offset == Force) ? 3DSTATE_SBE::Vertex URB Entry Read Offset: 3DSTATE_GS::GS_Enable ? 3DSTATE_GS:: Vertex URB Entry Output Read Offset: 3DSTATE_DS:DS_Enable ? 3DSTATE_DS:: Vertex URB Entry Output Read Offset : 3DSTATE_VS:: Vertex URB Entry Output Read Offset

Signal	SBE_INT::PrimId_override
Description	When true indicates that SBE provides the Primitive ID.
Formula	= 3DSTATE_GS::GS_Enable ? false : 3DSTATE_SBE::Primitive ID Override Component Select !=0

Barycentric Attribute Interpolation

Given hardware clipper and setup, some of the previous flexibility in the algorithm used to interpolate attributes is no longer available. Hardware uses barycentric parameters to aid in attribute interpolation, and these parameters are computed in hardware per-pixel (or per-sample) and delivered in the thread payload to the pixel shader. Also delivered in the payload are a set of vertex deltas (a0, a1, and a2) per channel of each attribute.

There are six different barycentric parameters that can be enabled for delivery in the pixel shader payload. These are enabled via the **Barycentric Interpolation Mode** bits in 3DSTATE_WM.

In the pixel shader kernel, the following computation is done for each attribute channel of each pixel/sample given the corresponding attribute channel a0/a1/a2 and the pixel/sample's b1/b2 barycentric parameters, where A is the value of the attribute channel at that pixel/sample:

A = a0 + (a1 * b1) + (a2 * b2)

Depth Offset

The state for depth offset in 3DSTATE_SF controls the depth offset function. Since this function was previously contained in the Windower stage, refer to the "Depth Offset" section in the Windower chapter for more details on this function.

Other SF Functions

The only other SF-related function is statistics gathering.

Statistics Gathering

The SF stage itself does not have any associated pipeline statistics; however, it counts the number of objects being output by the clipper on the clipper's behalf, since it is less feasible to have the CLIP unit figure out how many objects have been output by a clip thread. It is easy for the SF unit to count the number of objects it receives from the CLIP stage since it is decomposing the output primitive topologies into objects anyway.

If the **Statistics Enable** bit is set in SF_STATE, then SF will increment the CL_PRIMITIVES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) once for each object in each primitive topology it receives from the CLIP stage. This bit should always be set if clipping is enabled and pipeline statistics are desired.

Software should always clear the **Statistics Enable** bit in SF_STATE if the clipper is disabled since objects SF receives are not considered "primitives output by the clipper" unless the clipper is enabled. Note that the clipper can be disabled either using bypass mode via a PIPELINE_STATE_POINTERS command with **Clip Enable** clear *or* by setting **Clip Mode** in CLIP_STATE to CLIPMODE_ACCEPT_ALL.

Windower (WM) Stage

Overview

As mentioned in the *SF Unit* chapter, the SF stage prepares an object for scan conversion by the Window/Masker (WM) unit Refer to the *SF Unit* chapter for details on the screen-space geometry of objects to be rendered The WM unit uses the parameters provided by the SF unit in the object-specific rasterization algorithms.

The WM stage of the 3D pipeline performs the following operations (at a high level)

- Pre-scan-conversion modification of some primitive attributes, including
    - Application of Depth Offset to the position Z attribute
- Scan-conversion of the various primitive types, including
    - 2D clipping to the scissor/draw rectangle intersection
- Spawning of Pixel Shader (PS) threads to process the pixels resulting from scan-conversion

The spawned Pixel Shader (PS) threads are responsible for the following (high-level) operations

- interpolation of vertex attributes (other than X,Y,Z) to the pixel location
- performing any "Pixel Shader" operations dictated by the API PS program
    - Using the Sampler shared function to sample data from "texture" surfaces
    - Using the DataPort to perform general memory I/O
- Submitting the shaded pixel results to the DataPort for any subsequent "blending" (aka Output Merger) operation and write to the RenderCache.

The WM unit keeps a scoreboard of pixels being processed in outstanding PS threads in order to guarantee in-order rasterization results This allows the WM unit to overlap processing of several objects.

Inputs from SF to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object

- Handle of the Primitive URB Entry (PUE) that was written by the SF (Setup) thread This handle will be passed to all WM (PS) threads spawned from the WM's rasterization process.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.)
- Edge or line interpolation information (e.g., edge equation coefficients, etc.)
- Information on where the WM is to start rasterization of the object
- Object orientation (front/back-facing)
- Last Pixel indication (for line drawing)

Windower Pipelined State

3DSTATE_WM

The following inline state packets define the state used by the windower stage for different generations.

State Packets
3DSTATE_WM

Programming Note	
Context:	Windower Pipelined State
Note: WM Unit also receives 3DSTATE_WM_HZ_OP, 3DSTATE_RASTER, 3DSTATE_MULTISAMPLE, 3DSTATE_WM_CHROMAKEY, 3DSTATE_PS_BLEND, and 3DSTATE_PS_EXTRA	

Signal	WM_INT::ThreadDispatchEnable
Description	This bit, if set, indicates that it is possible for a PS thread to modify a render target.
Formula	= (3DSTATE_WM::ForceThreadDispatch == ON) || ((3DSTATE_WM::ForceThreadDispatch != OFF) && ! WM_INT::WM_HZ_OP && 3DSTATE_PS_EXTRA::PixelShaderValid && ((!3DSTATE_PS_EXTRA::PixelShaderDoesNotWriteRT && 3DSTATE_PS_BLEND::HasWriteableRT) || (3DSTATE_PS_EXTRA::PixelShaderHasUAV) || WM_INT:: Pixel Shader Kill Pixel || (WM_INT::Pixel Shader Computed Depth Mode != PSCDEPTH_OFF &&

	(WM_INT::Depth Test Enable || WM_INT::Depth Write Enable)) || (3DSTATE_PS_EXTRA::Computed Stencil && WM_INT::Stencil Test Enable) || (3DSTATE_WM::EDSC_Mode == 1 && (WM_INT::Depth Test Enable || WM_INT::Depth Write Enable || WM_INT::Stencil Test Enable)) || (WM_INT::RT Independent Rasterization Enable)))

Signal	WM_INT::Pixel Shader Computed Depth Mode
Description	This field specifies the computed depth mode for the pixel shader.
Formula	= (3DSTATE_PS_EXTRA::ForceComputedDepth == Force) ? 3DSTATE_PS_EXTRA::Pixel Shader Computed Depth Mode : (WM_INT::WM_HZ_OP || WM_INT::RT Independent Rasterization Enable) ? PSCDEPTH_OFF: 3DSTATE_PS_EXTRA::Pixel Shader Computed Depth Mode

Signal	WM_INT:: Pixel Shader Computed Stencil
Description	This field specifies the computed stencil mode for the pixel shader.
Formula	= (WM_INT::WM_HZ_OP) ? 0 : 3DSTATE_PS_EXTRA::Computed Stencil

Signal	WM_INT::Pixel Shader Uses Source Depth
Description	This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be passed in the payload.
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Source Depth

Signal	WM_INT::Pixel Shader Uses Source W

Description	This bit, if ENABLED, indicates that the PS kernel requires the interpolated source W value (vPos.w) to be passed in the payload
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Source W

Signal	WM_INT::Pixel Shader Uses Input Coverage Mask
Description	This bit, if ENABLED, indicates that the PS kernel requires the input coverage mask to be passed in the payload.
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Input Coverage Mask

Signal	WM_INT::Multisample Rasterization Mode
Description	This field determines whether multisample rasterization is enabled and how pixel sample points are defined.
Formula	See Table below: WM_INT::Multisample Rasterization Mode

WM_INT::Multisample Rasterization Mode

3DSTATE_RASTER :: Force Multisampling	Force	Force	Force	Force	Normal	Normal	Normal
3DSTATE_RASTER :: DX Multisample Rasterization Mode	MSRASTMODE_OFF_PIXEL	MSRASTMODE_OFF_PATTERN	MSRASTMODE_ON_PIXEL	MSRASTMODE_ON_PATTERN	*	*	*
WM_INT::WM_HZ_OP	*	*	*	*	True	True	False
3DSTATE_WM_HZ_OP:: Number of Multisamples	*	*	*	*	> NUMSAMPLES_1	NUMSAMPLES_1	*
WM_INT::Multisample Rasterization Mode	OFF_PIXEL	OFF_PATTERN	ON_PIXEL	ON_PATTERN	ON_PATTERN	ON_PIXEL	Determined from Table 1 in 3D Pipeline Windower

							> Multisampling > Multisample Modes/State)

Note: OFF_PIXEL, OFF_PATTERN, ON_PIXEL, ON_PATTERN modes are described in 3D Pipeline Windower > Multisampling > Multisample Modes/State.

Signal	WM_INT::Multisample Dispatch Mode
Description	This bit, determines how PS threads are dispatched
Formula	= (WM_INT::RT Independent Rasterization Enable)? PerPixel: (3DSTATE_PS_EXTRA::PixelShaderIsPerSample) ? PerSample : PerPixel

Signal	WM_INT::Pixel Shader Kill Pixel
Description	This bit, if ENABLED, indicates that the PS kernel or color calculator has the ability to kill (discard) pixels or samples, *other than due to depth or stencil testing*.
Formula	= (3DSTATE_WM::ForceKillPixel == ON) || ((3DSTATE_WM::ForceKillPixel != Off) && ! WM_INT::WM_HZ_OP && ! 3DSTATE_WM::EDSC_Mode == 2 && (WM_INT::Depth Write Enable || WM_INT::Stencil Write Enable) && (3DSTATE_PS_EXTRA::PixelShaderKillsPixels || 3DSTATE_PS_EXTRA:: **oMask Present to RenderTarget** || 3DSTATE_PS_BLEND::AlphaToCoverageEnable || 3DSTATE_PS_BLEND::AlphaTestEnable || 3DSTATE_WM_CHROMAKEY::ChromaKeyKillEnable

	))

Signal	WM_INT::Early Depth/Stencil Control
Description	This field specifies the behavior of early depth/stencil test.
Formula	= (WM_INT::WM_HZ_OP) ? EDSC_NORMAL : WM_INT::RT Independent Rasterization Enable ? EDSC_PSEXEC : 3DSTATE_WM::Early Depth/Stencil Control

Signal	WM_INT::RT Independent Rasterization Enable
Description	Enables Render Target Independent Rasterization.
Formula	= (WM_INT::WM_HZ_OP ? Disable : (3DSTATE_RASTER::ForceSampleCount != NUMRASTSAMPLES_0) ? Enable : Disable

Signal	WM_INT::Statistics Enable
Description	Enables Statistics
Formula	= (WM_INT::WM_HZ_OP) ? Disable : 3DSTATE_WM:: Statistics Enable

Signal	WM_INT::Polygon Stipple Enable
Description	Enables Poly Stipple
Formula	= (WM_INT::WM_HZ_OP) ? Disable : 3DSTATE_WM::Polygon Stipple Enable

Signal	WM_INT::WM_HZ_OP

Description	Enables WM_HZ_OP
Formula	= (3DSTATE_WM_HZ_OP::DepthBufferClear || 3DSTATE_WM_HZ_OP::DepthBufferResolve || 3DSTATE_WM_HZ_OP::Hierarchical Depth Buffer Resolve Enable || 3DSTATE_WM_HZ_OP::StencilBufferClear) ? Enable : Disable

Signal	WM_INT:: Pixel Location
Description	Sets the input pixel location to Center if UL and doing multisampling
Formula	(3DSTATE_MULTISAMPLE::Pixel Location && 3DSTATE_MULTISAMPLE::Pixel Position Offset Enable && WM_MULTISAMPLE_INT::Number of Multisamples > 0) ? 0 : 3DSTATE_MULTISAMPLE::Pixel Location

3DSTATE_SAMPLE_MASK

The following inline state packets define the sample mask state used by the windower stage for different generations.

State Packets
3DSTATE_SAMPLE_MASK

Signal	WM_INT:: Sample Mask Enable
Description	Sets Sample Mask used in rasterization

Formula	Switch(WM_MULTISAMPLE_INT::Number of Multisamples
	{
	Case NUMSAMPLES_1: WM_INT:: Sample Mask Enable = 0x0001; break;
	Case NUMSAMPLES_2: WM_INT:: Sample Mask Enable = 0x0003; break;
	Case NUMSAMPLES_4: WM_INT:: Sample Mask Enable = 0x000F; break;
	Case NUMSAMPLES_8: WM_INT:: Sample Mask Enable = 0x00FF; break;
	Add this additional case:
	Case NUMSAMPLES_16: WM_INT:: Sample Mask Enable = 0xFFFF; break;
	}

Signal	WM_INT:: Sample Mask
Description	Sets Sample Mask used in rasterization
Formula	= WM_INT:: Sample Mask Enable & (WM_INT::WM_HZ_OP) ? 3DSTATE_WM_HZ_OP::Sample Mask: 3DSTATE_SAMPLE_MASK::Sample Mask)

3DSTATE_WM_CHROMAKEY

3DSTATE_WM_HZ_OP

State Restrictions

State	Restriction
3DSTATE_PS::Render Target Fast Clear Enable	Must be disabled
3DSTATE_PS:: Render Target Resolve Enable	Must be disabled
3DSTATE_WM:: Legacy Depth Buffer Clear	Must be disabled
3DSTATE_WM:: Legacy Depth Buffer Resolve	Must be disabled
3DSTATE_WM:: Legacy Hierarchical Depth Buffer Resolve Enable	Must be disabled
3DSTATE_MULTISAMPLE::Pixel Location	Must be set according to the API being used.
3DSTATE_CLEAR_PARAMS ::Depth Clear Value	Must be programmed according to the API when 3DSTATE_WM_HZ_OP::Depth Buffer Clear is set
3DSTATE_CLEAR_PARAMS ::Depth Clear Value Valid	Must be enabled when 3DSTATE_WM_HZ_OP::Depth Buffer Clear is set

State Overrides

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
SF_INT:: Statistics Enable	Disable	Disable	Disable	Disable
SF_INT:: View Transform Enable	Disable	Disable	Disable	Disable
SF_INT::Multisample Rasterization Mode	(3DSTATE_WM_HZ_OP ::NumberOfSamples > 1) ?	(3DSTATE_WM_HZ_OP ::NumberOfSamples > 1) ?	(3DSTATE_WM_HZ_OP ::NumberOfSamples > 1) ?	(3DSTATE_WM_HZ_OP ::NumberOfSamples > 1) ?

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
	ON_PATTERN : ON_PIXEL	ON_PATTERN : ON_PIXEL	ON_PATTERN : ON_PIXEL	ON_PATTERN : ON_PIXEL
SF_INT::Cull Mode	CULLMODE_BACK	CULLMODE_BACK	CULLMODE_BACK	CULLMODE_BACK
SF_INT::Scissor Rectangle Enable	3DSTATE_WM_HZ_OP::Scissor Rectangle Enable	3DSTATE_WM_HZ_OP::Scissor Rectangle Enable	3DSTATE_WM_HZ_OP::Scissor Rectangle Enable	3DSTATE_WM_HZ_OP::Scissor Rectangle Enable
SF_INT::RT Independent Rasterization Enable	Disable	Disable	Disable	Disable
SF_INT::FrontFace Fill Mode	SOLID	SOLID	SOLID	SOLID
SF_INT::FrontWinding	FRONTWINDING_CW	FRONTWINDING_CW	FRONTWINDING_CW	FRONTWINDING_CW
SF_INT::Render Target Array index	0	0	0	0
SF_INT::Viewport index	0	0	0	0
SF_INT:: Geometry Hashing Disable	Disable	Disable	Disable	Disable
WM_INT::StencilTestEnable	Enable	Stencil buffer Clear ? Enable : Disable	Disable	Disable
WM_INT::StencilWriteEnable	Enable	Stencil buffer Clear ? Enable : Disable	Disable	Disable
WM_INT::DepthTestEnable	Disable	Disable	Enable	Disable
WM_INT::DepthWriteEnable	Depth buffer Clear ? Enable : Disable	Enable	Enable	Enable
WM_INT::DepthTestFunction	NEVER	NEVER	NEVER	NEVER

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
WM_INT::StencilTestFunction	ALWAYS	Stencil buffer Clear ? ALWAYS: No Override	No Override	No Override
WM_INT::StencilPassDepthPassOp	REPLACE	Stencil buffer Clear ? REPLACE: No Override	No Override	No Override
WM_INT:: Statistics Enable	Disable	Disable	Disable	Disable
WM_INT::ThreadDispatchEnable	Disable	Disable	Disable	Disable
WM_INT:: Pixel Shader Kill Pixel	Disable	Disable	Disable	Disable
WM_INT:: Pixel Shader Computed Depth Mode	PSCDEPTH_OFF	PSCDEPTH_OFF	PSCDEPTH_OFF	PSCDEPTH_OFF
: WM_INT: Computed Stencil Enable	Disable	Disable	Disable	Disable
WM_INT::RT Independent Rasterization Enable	Disable	Disable	Disable	Disable
WM_INT::Polygon Stipple Enable	Disable	Disable	Disable	Disable
WM_INT::Multisample Rasterization Mode	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)
MULTISAMPLE_INT::Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples
WM_INT::Sample Mask	3DSTATE_WM_HZ_OP:: Sample Mask	3DSTATE_WM_HZ_OP:: Sample Mask	3DSTATE_WM_HZ_OP:: Sample Mask	3DSTATE_WM_HZ_OP:: Sample Mask
WM_INT::Early Depth/Stencil Control	EDSC_NORMAL	EDSC_NORMAL	EDSC_NORMAL	EDSC_NORMAL
WM_INT:: Full	Depth buffer clear ?	WM_HZ_OP:: Full	Disable	Disable

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
Surface Depth Clear	WM_HZ_OP:: Full Surface Depth Clear : Disable	Surface Depth Clear		
WM_INT:: Full Surface Depth Clear	Depth buffer clear ? WM_HZ_OP:: Full Surface Depth Clear : Disable	WM_HZ_OP:: Full Surface Depth Clear	Disable	Disable

3DSTATE_WM_DEPTH_STENCIL

Signal	WM_INT::StencilWriteEnable
Description	Enables writes to the Stencil Buffer
Formula	= 3DSTATE_STENCIL_BUFFER::STENCIL_BUFFER_ENABLE && 3DSTATE_DEPTH_BUFFER::STENCIL_WRITE_ENABLE && ((WM_INT::WM_HZ_OP ? Use the WM_INT::StencilWriteEnable from WM_HZ_OP table : WM_INT::StencilTestEnable && 3DSTATE_WM_DEPTH_STENCIL::StencilBufferWriteEnable))

Signal	WM_INT::StencilTestEnable
Description	Enables Stencil Test
Formula	= 3DSTATE_STENCIL_BUFFER::STENCIL_BUFFER_ENABLE && (WM_INT::WM_HZ_OP ? Use the WM_INT::StencilTestEnable from WM_HZ_OP table : (3DSTATE_WM_DEPTH_STENCIL::StencilTestEnable && !WM_INT::RT Independent Rasterization Enable))

Signal	WM_INT::DepthTestEnable
Description	Enables Depth Test

Formula	= (3DSTATE_DEPTH_BUFFER::SURFACE_TYPE != NULL) && (WM_INT::WM_HZ_OP ? Use the WM_INT::DepthTestEnable from WM_HZ_OP table : (3DSTATE_WM_DEPTH_STENCIL::DepthTestEnable && ! WM_INT::RT Independent Rasterization Enable))

Signal	WM_INT::DepthWriteEnable
Description	Enables Depth Write
Formula	= (3DSTATE_DEPTH_BUFFER::SURFACE_TYPE != NULL) && 3DSTATE_DEPTH_BUFFER::DEPTH_WRITE_ENABLE && (WM_INT::WM_HZ_OP ? Use the WM_INT::DepthWriteEnable from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::DepthWriteEnable)

Signal	WM_INT::DepthTestFunction
Description	Depth Test Function
Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::DepthTestFunction from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::DepthTestFunction

Signal	WM_INT::StencilTestFunction
Description	Stencil Test Function
Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::StencilTestFunction from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::StencilTestFunction

Signal	WM_INT::StencilPassDepthPassOp
Description	StencilPassDepthPassOp

Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::StencilPassDepthPassOp from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::StencilPassDepthPassOp

Signal	WM_INT::Stencil Test Mask
Description	Stencil test Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Stencil Test Mask

Signal	WM_INT::Stencil Write Mask
Description	Stencil Write Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Stencil Write Mask

Signal	WM_INT::BackFace Stencil Test Mask
Description	Stencil test Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL:: Backface Stencil Test Mask

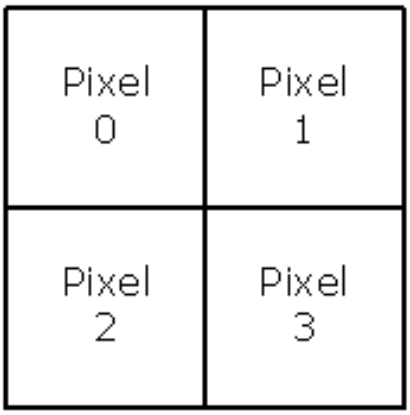
Signal	WM_INT:: BackFace Stencil Write Mask
Description	Stencil Write Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Backface Stencil Write Mask

Rasterization

The WM unit uses the setup computations performed by the SF unit to rasterize objects into the corresponding set of pixels Most of the controls regarding the screen-space geometry of rendered objects are programmed via the SF unit.

The rasterization process generates pixels in 2x2 groups of pixels called *subspans* (see *Pixels with a SubSpan* below) which, after being subjected to various inclusion/discard tests, are grouped and passed to spawned Pixel Shader (PS) threads for subsequent processing Once these PS threads are spawned, the WM unit provides only bookkeeping functions on the pixels Note that the WM unit can proceed on to rasterize subsequent objects while PS threads from previous objects are still executing.

Pixels with a SubSpan



B6850-01

Drawing Rectangle Clipping

The Drawing Rectangle defines the maximum extent of pixels which can be rendered Portions of objects falling outside the Drawing Rectangle will be clipped (pixels discarded) Implementations will typically discard objects falling completely outside of the Drawing Rectangle as early in the pipeline as possible There is no control to turn off Drawing Rectangle clipping – it is unconditional.

For the purposes of clipping, the Drawing Rectangle must itself be clipped to the destination buffer extents (The Drawing Rectangle Origin, used to offset relative X,Y coordinates earlier in the pipeline, is permitted to lie offscreen). The **Clipped Drawing Rectangle X,Y Min,Max** state variables (programmed via 3DSTATE_DRAWING_RECTANGLE – See *SF Unit*) defines the intersection of the Drawing Rectangle and the Color Buffer It is specified with non-negative integer pixel coordinates relative to the Destination Buffer upper-left origin.

Pixels with coordinates outside of the Drawing Rectangle cannot be rendered (i.e., the rectangle is inclusive) For example, to render to a full-screen 1280x1024 buffer, the following values would be required: Xmin=0, Ymin=0, Xmax=1279 and Ymax=1023

For “full screen” rendering, the Drawing Rectangle coincides with the screen-sized buffer For “front-buffer windowed” rendering it coincides with the destination “window”.

Line Rasterization

See *SF Unit* chapter for details on the screen-space geometry of the various line types.

Coverage Values for Anti-Aliased Lines

The WM unit is provided with both the **Line Anti-Aliasing Region Width** and **Line End Cap Anti-aliasing Region Width** state variables (in WM_STATE) in order to compute the coverage values for anti-aliased lines.

3DSTATE_AA_LINE_PARAMS

3DSTATE_AA_LINE_PARAMETERS

The slope and bias values should be computed to closely match the reference rasterizer results Based on empirical data, the following recommendations are offered:

The final alpha for the center of the line needs to be 148 to match the reference rasterizer In this case, the Lo to edge 0 and edge 3 will be the same Since the alpha for each edge is multiplied together, we get:

edge0alpha * edge1alpha = 148/255 = 0.580392157

Since edge0alpha = edge3alpha we get:

$(edge0alpha)^2 = 0.580392157$

edge0alpha = sqrt(0.580392157) = 0.761834731 at the center pixel

The desired alpha for pixel 1 = 54/255 = 0.211764706

The slope is (0.761834731 – 0.211764706) = 0.550070025

Since we are using 8 bit precision, the slope becomes

AA Coverage [EndCap] Slope = 0.55078125

The alpha value for Lo = 0 (second pixel from center) determines the bias term and is equal to

(0.211764706 – 0.550070025) = -0.338305319

With 8 bits of precision the programmed bias value

Line Stipple

Line stipple, controlled via the **Line Stipple Enable** state variable in WM_STATE, discards certain pixels that are produced by non-AA line rasterization.

The line stipple rule is specified via the following state variables programmed via 3DSTATE_LINE_STIPPLE: the 16-bit **Line Stipple Pattern** (p), **Line Stipple Repeat Count** l, and **Line Stipple Inverse Repeat Count.** Sofware must compute **Line Stipple Inverse Repeat Count** as 1.0f / **Line Stipple Repeat Count** and then converted from float to the required fixed point encoding (see 3STATE_LINE_STIPPLE).

The WM unit maintains an internal Line Stipple Counter state variable (s) The initial value of s is zero; s is incremented after production of each pixel of a line segment (pixels are produced in order, beginning at the starting point and working towards the ending point). S is reset to 0 whenever a new primitive is

processed (unless the primitive type is LINESTRIP_CONT or LINESTRIP_CONT_BF), and before every line segment in a group of independent segments (LINELIST primitive).

During the rasterization of lines, the WM unit computes:

$$b = \lfloor s/r \rfloor \bmod 16,$$

A pixel is rendered if the bth bit of p is 1, otherwise it is discarded. The bits of p are numbered with 0 being the least significant and 15 being the most significant.

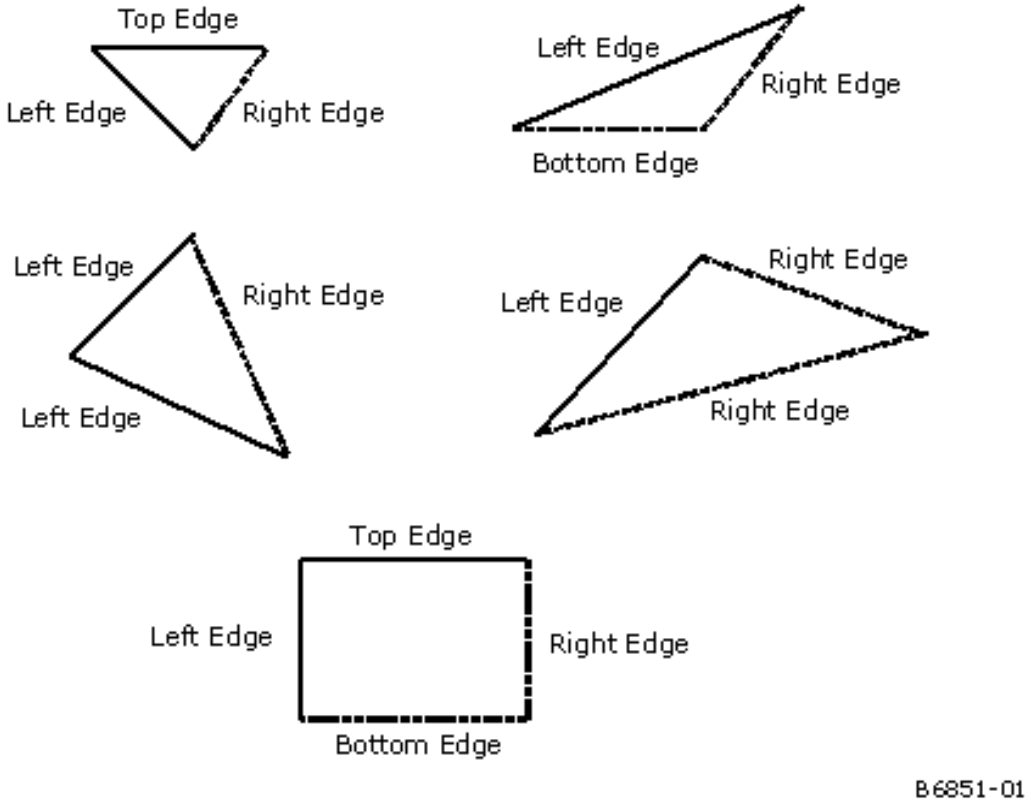
3DSTATE_LINE_STIPPLE

Polygon (Triangle and Rectangle) Rasterization

The rasterization of LINE, TRIANGLE, and RECTANGLE objects into pixels requires a "pixel sampling grid" to be defined This grid is defined as an axis-aligned array of pixel sample points spaced exactly 1 pixel unit apart If a sample point falls within one of these objects, the pixel associated with the sample point is considered "inside" the object, and information for that pixel is generated and passed down the pipeline

For TRIANGLE and RECTANGLE objects, if a sample point intersects an edge of the object, the associated pixel is considered "inside" the object if the intersecting edge is a "left" or "top" edge (or, more exactly, the intersected edge is not a "right" or "bottom" edge) Note that "top" and "bottom" edges are by definition exactly horizontal. See *TRIANGLE and RECTANGLE Edge Types* below for the edge types for representative TRIANGLE and RECTANGLE objects (solid edges are inclusive, dashed edges are exclusive).

TRIANGLE and RECTANGLE Edge Types



Polygon Stipple

The *Polygon Stipple* function, controlled via the **Polygon Stipple Enable** state variable in WM_STATE, allows only selected pixels of a repeated 32x32 pixel pattern to be rendered Polygon stipple is applied only to the following primitive types:

3DPRIM_POLYGON
3DPRIM_TRIFAN
3DPRIM_TRILIST
3DPRIM_TRISTRIP
3DPRIM_TRISTRIP_REVERSE

Note that the 3DPRIM_TRIFAN_NOSTIPPLE object is never subject to polygon stipple.

The stipple pattern is defined as a 32x32 bit pixel mask via the 3DSTATE_POLY_STIPPLE_PATTERN command. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

The origin of the pattern is specified via **Polygon Stipple X,Y Offset** state variables programmed via the 3DSTATE_POLY_STIPPLE_OFFSET command The offsets are pixel offsets from the Color Buffer origin to the upper left corner of the stipple pattern. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_PATTERN

Multisampling

The multisampling function has two components:

- **Multisample Rasterization**: multisample rasterization occurs at a subpixel level, wherein each pixel consists of a number of "samples" at state-defined positions within the pixel footprint. Coverage of the primitive as well as color calculator operations (stencil test, depth test, color buffer blending, etc.) are done at the sample level. In addition the pixel shader itself can optionally run at the sample level depending on a separate state field.
- **Multisample Render Targets (MSRT)**: The render targets, as well as the depth and stencil buffers, now have the ability to store per-sample values. When combined with multisample rasterization, color calculator operations such as stencil test, depth test, and color buffer blending are done with the destination surface containing potentially different values per sample.

3DSTATE_MULTISAMPLE

Signal	WM_MULTISAMPLE_INT::Number of Multisamples
Description	Set the number of multisamples
Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP::Number of Multisamples : 3DSTATE_MULTISAMPLE::Number of Multisamples

3DSTATE_SAMPLE_PATTERN

Multisample ModesState

A number of state variables control the operation of the multisampling function. The following table indicates the states and their location. Refer to the state definition for more details.

State Element	Source	Description
Multisample Rasterization Mode	WM_INT::Multisample Rasterization Mode	Controls whether rasterization of non-lines is performed on a pixel or sample basis (PIXEL vs. PATTERN), and whether multisample rasterization of lines is enabled (OFF vs. ON). From this generation forward, this state element becomes an internal signal computed by other state variables (also listed here) unless certain modes are set, which can be seen in the WM_INT equation for the signal.
Multisample Dispatch Mode	WM_INT::Multisample Dispatch Mode	Controls whether the pixel shader is executed per pixel or per sample.
Number of Multisamples	3DSTATE_MULTISAMPLE and SURFACE_STATE	Indicates the number of samples per pixel contained on the surface. This field in 3DSTATE_MULTISAMPLE must match the corresponding field in SURFACE_STATE for each render target. The depth, hierarchical depth, and stencil buffers inherit this field from 3DSTATE_MULTISAMPLE.
RTIR Enabled	3DSTATE_RASTER::ForcedSampleCount != NUMRASTSAMPLES_0	Enable Render Target Independent Rasterization.
Pixel Location	3DSTATE_MULTISAMPLE	Indicates the subpixel location where values specified as “pixel” are sampled. This is either the upper left corner or the center.
MSAA Sample Offsets	3DSTATE_SAMPLE_PATTERN	For each of the N samples, specifies the subpixel location of each sample.
RTIR Sample Offsets	3DSTATE_SAMPLE_PATTERN	For each of the N samples, specifies the subpixel location of each sample.
API Mode	3DSTATE_RASTER	One of the deciding factors of what the Multisample Rasterization Mode should be according to WM_INT::Multisample Rasterization Mode. Software

State Element	Source	Description
		sets this field according to the API's version.
DX Multisample Rasterization Enable	3DSTATE_RASTER	Controls ON/OFF part of Multisample Rasterization Mode, depending on the API Mode according to WM_INT::Multisample Rasterization Mode.

This table does not apply if (3DSTATE_RASTER::ForceMultisampleRasterMode == Force) or (WM_INT::WM_HZ_OP == true).

Table 1: Multisample Rasterization Modes

Number of Multisamples	NUMSAMPLES_1	NUMSAMPLES_1	> NUMSAMPLES_1	> NUMSAMPLES_1	Any	Any	Any	Any
DX Multisample Rasterization Enable	0	1	0	1	0	1	0	1
Rast Number of Samples	Disabled	Disabled	Disabled	Disabled	NUMRAST SAMPLES_1	NUMRAST SAMPLES_1	> NUMRAST SAMPLES_1	> NUMRAST SAMPLES_1
API Mode == DX9.0/OGL	OFF_PIXEL	OFF_PIXEL	OFF_PIXEL	ON_PATTERN	Invalid	Invalid	Invalid	Invalid
API Mode == DX10.0	OFF_PIXEL	ON_PIXEL	OFF_PIXEL	ON_PATTERN	OFF_PIXEL	Invalid	Invalid	Invalid
API Mode == DX10.1+	OFF_PIXEL	ON_PIXEL	OFF_PATTERN	ON_PATTERN	OFF_PIXEL	ON_PIXEL	OFF_PATTERN	ON_PATTERN

Definitions for lines terms used in Table 2 through Table 4:

- **Legacy Lines:** Way of drawing lines that allows Diamond Lines (SF_STATE::Line Width == 0.0), Non-anti-aliased Wide Lines (SF_STATE::Line Width != 0.0), and Line Stippling (3DSTATE_WM:: Line Stipple Enable == 1).
- **AA Lines:** Way of drawing lines that allows Anti-aliased line. These are lines rendered as rectangles that are

centered on, and aligned to, the line joining the endpoint vertices with coverage value (referred to as Anti-alias Alpha) computed per pixel.

AA Line Support Requirement
SF_INT::Anti-aliasing Enable == 1

- **MSAA Lines:** Way of drawing lines that allows Multisample Anti-aliased lines. These are lines rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices, but no Anti alias alpha coverage is computed.

Table 2: Type of Line Algorithm Given an Arrangement of State Variables

Multisample Rasterization Mode	Anti-Aliasing Enable	SF_STATE::Line Width	Line Algorithm
OFF_*	0	Non-Zero	Non-Anti-aliased Wide Lines
OFF_*	0	0.0	Diamond Lines
OFF_*	1	Non-Zero	See Note A below.
OFF_*	1	0.0	Diamond Lines
ON_*	*	*	MSAA Lines

Note A: Anti-Aliasing Details for Table 2

Anti-Aliasing Details
Anti-Aliased Lines with Alpha Coverage

Table 3: Multisample Modes with RTIR Disabled

Number of Multisamples	MS RAST MODE	MS DISP MODE	HW Mode
NUMSAMPLES_1	OFF_PIXEL	PERSAMPLE	**Legacy Non-MSAA Mode** 1X rasterization, using Pixel Location Legacy lines or AA-line rasterization 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PIXEL	PERSAMPLE	**1X Multisampling Mode** 1X rasterization, using Pixel Location MSAA lines only, using Pixel Location 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	-	PERPIXEL	Treated the same as PERSAMPLE

	ON_PATTERN	-	Invalid
	OFF_PATTERN	-	Invalid
n where n > NUMSAMPLES_1	OFF_PIXEL	PERPIXEL	**MSRT Only, PerPixel PS** 1X rasterization, using Pixel Location See Note B below. 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**MSRT Only, PerSample PS** 1X rasterization, using Pixel Location See Note B below. nX PS, all samples at Pixel Location nX output merge, eval Depth at Pixel Location
	ON_PIXEL	PERPIXEL	**Multibuffering MSAA, PerPixel PS** 1X rasterization, using Pixel Location MSAA lines only 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**Multibuffering MSAA, PerSample PS** 1X rasterization, using Pixel Location MSAA lines only nX PS, all samples at Pixel Location nX output merge, eval Depth at Pixel Location
	OFF_PATTERN	PERPIXEL	**Mixed Mode, PerPixel PS** See Note B below. Non-Lines: nX rasterization, using Sample Offsets 1X PS, sample at Pixel Location nX output merge, eval depth at Sample Offsets
		PERSAMPLE	**Mixed Mode, PerSample PS** See Note B below. Non-Lines: nX rasterization, using Sample Offsets

			nX PS, sample at Pixel Location or Sample Offsets nX output merge, eval depth at Sample Offsets
	ON_PATTERN	PERPIXEL	**Pattern MSAA, PerPixel PS** nX rasterization, using Sample Offsets MSAA lines only 1X PS, sample at Pixel Location nX output merge, eval depth at Sample Offsets
		PERSAMPLE	**Pattern MSAA, PerSample PS** nX rasterization, using Sample Offsets MSAA lines only nX PS, sample at Pixel Location or Sample Offsets nX output merge, eval depth at Sample Offsets

Note B: Line Details for Table 3 and Table 4

Line Details
Legacy lines or AA-line rasterization. For PERPIXEL or PERSAMPLE in Table 3 use pixel location. For OFF_PATTERN in Table 4 use pixel location.

Table 4: Multisample Modes with RTIR Enabled

Rast Number of Samples	MS RAST MODE	HW Mode
NUMRASTSAMPLES_1	OFF_PIXEL	**Legacy Non-MSAA Mode** 1X rasterization, using Pixel Location Legacy lines or AA-line rasterization 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PIXEL	**1X Multisampling Mode** 1X rasterization, using Pixel Location MSAA lines only, using Pixel Location 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PATTERN	Invalid

	OFF_PATTERN	Invalid
n where n > NUMRASTSAMPLES_1	OFF_PIXEL	**Invalid**
	ON_PIXEL	**Invalid**
	OFF_PATTERN	**Mixed Mode, PerPixel PS** See Note B above. Non-Lines: nX rasterization, using Sample Offsets 1X PS, sample at Pixel Location 1X output merge, eval depth atPixel Location
	ON_PATTERN	**Pattern RTIR, PerPixel PS** nX rasterization, using Sample Offsets MSAA lines only 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location

Note: Multisample Dispatch Mode is not taken into account in Table 4 given that with RTIR:

Details
The value of PERSAMPLE is converted to PERPIXEL internally.

Other WM Functions

The only other WM function is Statistics Gathering.

Statistics Gathering

If **Statistics Enable** is set in WM_STATE or 3DSTATE_WM, the Windower increments the PS_INVOCATIONS_COUNT register once for each unmasked pixel (or sample) that is *dispatched* to a Pixel Shader thread.

If **Early Depth Test Enable** is set it is possible for pixels or samples to be discarded before reaching the Pixel Shader due to failing the depth or stencil test. PS_INVOCATIONS_COUNT will still be incremented for these pixels or samples since the depth test occurs after the pixel shader from the point of view of SW.

Pixel

This section contains the following subsections:

- **Depth and Stencil**, which covers the Depth and Stencil test functions
- **Pixel Dispatch**, which covers pixel shader state, pixel grouping, multisampling effects on pixel shader dispatch, and pixel shader thread payload

- **Pixel Backend**, which covers backend processing

Early Depth/Stencil Processing

The Windower/IZ unit provides the Early Depth Test function, a major performance-optimization feature where an attempt is made to remove pixels that fail the Depth and Stencil Tests prior to pixel shading. This requires the WM unit to perform the interpolation of pixel ("source") depth values, read the current ("destination") depth values from the cached depth buffer, and perform the Depth and Stencil Tests As the WM unit has per-pixel source and destination Z values, these values are passed in the PS thread payload, if required.

Depth Offset

Note: The depth offset function is contained in SF unit, thus the state to control it is also contained in SF unit.

There are occasions where the Z position of some objects need to be slightly offset to reduce artifacts due to coplanar or near-coplanar primitives. A typical example is drawing the edges of triangles as wireframes – the lines need to be drawn slightly closer to the viewer to ensure they will not be occluded by the underlying polygon. Another example is drawing objects on a wall – without a bias on the z positions, they might be fully or partially occluded by the wall.

The device supports *global* depth offset, applied only to triangles, that bases the offset on the object's z slope Note that there is no clamping applied at this stage after the Z position is offset – clamping to [0,1] can be performed later after the Z position is interpolated to the pixel. This is preferable to clamping prior to interpolation, as the clamping would change the Z slope of the entire object.

The Global Depth Offset function is controlled by the **Global Depth Offset Enable** state variable in WM_STATE Global Depth Offset is only applied to 3DOBJ_TRIANGLE objects.

When Global Depth Offset Enable is ENABLED, the pipeline will compute:

```
    MaxDepthSlope = max(abs(dZ/dX),abs(dz/dy)) // approximation of max depth slope for
polygon
```

When UNORM Depth Buffer is at Output Merger (or no Depth Buffer):

```
    Bias = GlobalDepthOffsetConstant * r + GlobalDepthOffsetScale * MaxDepthSlope
```

Where r is the minimum representable value > 0 in the depth buffer format, converted to float32 (note: If state bit **Legacy Global Depth Bias Enable** is set, the r term will be forced to 1.0)

When Floating Point Depth Buffer at Output Merger:

```
    Bias = GlobalDepthOffsetConstant * 2^(exponent(max z in primitive) – r) +
GlobalDepthOffsetScale * MaxDepthSlope
```

Where r is the # of mantissa bits in the floating point representation (excluding the hidden bit), e.g. 23 for float32 (note: If state bit Legacy Global Depth Bias Enable is set, no scaling is applied to the GobalDepthOffsetConstant).

Adding Bias to z:

```
if (GlobalDepthOffsetClamp > 0)
    Bias = min(DepthBiasClamp, Bias)
else if(GlobalDepthOffsetClamp < 0)
    Bias = max(DepthBiasClamp, Bias)
// else if GlobalDepthOffsetClamp == 0, no clamping occurs
z = z + Bias
```

Biasing is constant for a given primitive. The biasing formulas are performed with float32 arithmetic Global Depth Bias is not applied to any point or line primitives.

Early Depth Test/Stencil Test/Write

When **Early Depth Test Enable** is ENABLED, the WM unit will attempt to discard depth-occluded pixels during scan conversion (before processing them in the Pixel Shader). Pixels are only discarded when the WM unit can ensure that they would have no impact to the ColorBuffer or DepthBuffer. This function is therefore only a performance feature.

Note: For , the **Early Depth Test Enable** bit is no longer present. This function is always enabled.

If some pixels within a subspan are discarded, only the pixel mask is affected indicating that the discarded pixels are not active. If all pixels within a subspan are discarded, that subspan will not even be dispatched.

Software-Provided PS Kernel Info

For the WM unit to properly perform Early Depth Test and supply the proper information in the PS thread payload (and even determine if a PS thread needs to be dispatched), it requires information regarding the PS kernel operation This information is provided by a number of state bits in WM_STATE, as summarized in the following table.

State Bit	Description
Pixel Shader Kill Pixel	This must be set when there is a chance that valid pixels passed to a PS thread may be discarded. This includes the discard of pixels by the PS thread resulting from a “killpixel” or “alphatest” function or as dictated by the results of the sampling of a “chroma-keyed” texture The WM unit needs this information to prevent early depth/stencil writes for pixels which might be killed by the PS thread, etc. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Computed Depth	This must be set when the PS thread computes the “source” depth value (i.e., from the API POV, writes to the “oDepth” output) In this case the WM unit can’t make any decisions based on the WM-interpolated depth value. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Uses Source Depth	Must be set if the PS thread requires the WM-interpolated source depth value. This forces the source depth to be passed in the thread payload where otherwise the WM unit would not have seen it as required. See WM_STATE/3DSTATE_WM for more information.

Hierarchical Depth Buffer

A hierarchical depth buffer is supported to reduce memory traffic due to depth buffer accesses. This buffer is supported only in Tile Y memory.

The **Surface Type**, **Height**, **Width**, **Depth**, **Minimum Array Element**, **Render Target View Extent**, and **Depth Coordinate Offset X/Y** of the hierarchical depth buffer are inherited from the depth buffer. The height and width of the hierarchical depth buffer that must be allocated are computed by the following formulas, where HZ is the hierarchical depth buffer and Z is the depth buffer. The Z_Height, Z_Width, and Z_Depth values given in these formulas are those present in 3DSTATE_DEPTH_BUFFER incremented by one.

The Z_Height and Z_Width values must equal those present in 3DSTATE_DEPTH_BUFFER incremented by one.

Surface Type	HZ_Width (Bytes)	HZ_Height (Rows)	HZ_Qpitch (Rows)
SURFTYPE_1D	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * Z_Depth	see below
SURFTYPE_2D	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * Z_Depth	see below
SURFTYPE_3D			not applicable
SURFTYPE_CUBE	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * 6 * Z_Depth	see below

To compute the minimum QPitch for the HZ surface, the height of each LOD in pixels is determined using the equations for hL in the GPU Overview volume, using a vertical alignment j=8. The following equation gives the minimum HZ_QPitch based on largest LOD m defined in the surface:

$$HZ_QPitch = h_0 + max\left(h_1, \sum_{i=2}^{m} \mathrm{h_i}\right)$$

If m is less than 2, treat all h_L with L > m as zero and use the above equation.

The minimum HZ_Height required for a 3D surface must be computed based on h_L parameters documented in the GPU Overview volume, and the maximum LOD m:

The format of the data in the hierarchical depth buffer is not documented here, as this surface needs only to be allocated by software. Hardware will read and write this surface during operation and its contents are discarded once the last primitive is rendered that uses the hierarchical depth buffer.

The hierarchical depth buffer can be enabled whenever a depth buffer is defined, with its effect being invisible other than generally higher performance. The only exception is the hierarchical depth buffer must be disabled when using software tiled rendering.

If HiZ is enabled, you must initialize the clear value by either:

1. Perform a depth clear pass to initialize the clear value.
2. Send a 3dstate_clear_params packet with valid = 1.

Without one of these events, context switching will fail, as it will try to save off a clear value even though no valid clear value has been set. When context restore happens, HW will restore an uninitialized clear value.

Depth Buffer Clear

With the hierarchical depth buffer enabled, performance is generally improved by using the special clear mechanism described here to clear the hierarchical depth buffer and the depth buffer. This is enabled though the **Depth Buffer Clear** field in WM_STATE or 3DSTATE_WM or using the 3DSTATE_WM_HZ_OP. This bit can be used to clear the depth buffer in the following situations:

- Complete depth buffer clear.
- Partial depth buffer clear with the clear value the same as the one used on the previous clear.
- Partial depth buffer clear with the clear value different than the one used on the previous clear can use this mechanism if a depth buffer resolve is performed first.

The following is required when performing a depth buffer clear using any of the above clearing methods (WM_STATE, 3DSTATE_WM or 3DSTATE_WM_HZ_OP).

- The hierarchical depth buffer enable must be set in the 3DSTATE_DEPTH_BUFFER.
- The fields in 3DSTATE_CLEAR_PARAMS are set to indicate the source of the clear value and (if source is in this command) the clear value itself.
- The clear value must be between the min and max depth values (inclusive) defined in the CC_VIEWPORT. If the depth buffer.
    - The following alignment restrictions need to be met while doing the fast-clear:

Alignment Restriction
The minimum granularity of clear is one pixel, but all samples of the pixel must be cleared. Clearing partial samples of a pixel is not supported. If a newly allocated depth buffer is not padded to an integer multiple of 8x4 pixels, and if the first operation on the depth buffer does not clear the entire width and height of the surface, then first a HiZ ambiguate must be done on the portions of the depth buffer that are not cleared. If the depth buffer clear operation does clear the entire width and height of the surface, then the “full surface clear” bit in 3DSTATE_WM_OP must be set to 1.

The following is required when performing a depth buffer clear with using the WM_STATE or 3DSTATE_WM:

- If other rendering operations have preceded this clear, a PIPE_CONTROL with depth cache flush enabled, Depth Stall bit enabled must be issued before the rectangle primitive used for the depth buffer clear operation.
- **Depth Test Enable** must be disabled and **Depth Buffer Write Enable** must be enabled (if depth is being cleared).

- Stencil buffer clear can be performed at the same time by enabling Stencil Buffer Write Enable. Stencil Test Enable must be enabled and Stencil Pass Depth Pass Op set to REPLACE, and the clear value that is placed in the stencil buffer is the **Stencil Reference Value** from COLOR_CALC_STATE.
- Note also that stencil buffer clear can be performed without depth buffer clear. For stencil only clear, **Depth Test Enable** and **Depth Buffer Write Enable** must be disabled.

In some cases **Depth Buffer Clear** cannot be enabled and the legacy method of clearing must be used:

- If the depth buffer format is D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT.
- If stencil test is enabled but the separate stencil buffer is disabled.

Depth Buffer Clear Workaround

Depth buffer clear pass using any of the methods (WM_STATE, 3DSTATE_WM or 3DSTATE_WM_HZ_OP) must be followed by a PIPE_CONTROL command with DEPTH_STALL bit and Depth FLUSH bits "**set**" before starting to render. DepthStall and DepthFlush are not needed between consecutive depth clear passes nor is it required if the depth-clear pass was done with "full_surf_clear" bit set in the 3DSTATE_WM_HZ_OP.

Note: If using the optimized depth buffer clear, this pipecontrol should be done after the resetting of the clear/resolve bits in the 3DSTATE_WM_HZ_OP (step #8).

Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete If the depth buffer is retained and used for another purpose (i.e as input to the sampling engine as a shadow map), it must first be "resolved" This is done by setting the **Depth Buffer Resolve Enable** field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the nest rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

The following is required when performing a depth buffer resolve:

- The surface must have been initialized with a Depth Buffer Clear after its allocation to initialize the Depth Clear Value.
- A rectangle primitive of the same size as the previous depth buffer clear operation must be delivered, and depth buffer state cannot have changed since the previous depth buffer clear operation.
- **Depth Test Enable** must be enabled with the **Depth Test Function** set to NEVER. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

Programming Note	
Context:	HTML
HW uses the clear value from the 3DSTATE_CLEAR_PARAM. If you change the value in the 3DSTATE_CLEAR_PARAMS before resolve, it will flush the depth caches and have the new-clear value in its register. When doing the resolve pass, it is driver's responsibility to make sure that the clear-value for the depth buffer is the same one as the clear-pass.	

Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation If this occurs, the hierarchical depth buffer must be *resolved* to avoid incorrect device behavior. This is done by setting the Hierarchical Depth Buffer Resolve Enable field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be delivered.
- **Depth Test Enable** must be disabled. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel**, and **Pixel Shader Computed Depth** must all be disabled.

Optimized Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete If the depth buffer is retained and used for another purpose (i.e locked by the app), it must first be "resolved" This is done by setting the **Depth Buffer Resolve Enable** field in 3DSTATE_WM_HZ_OP. The depth buffer resolve uses the same sequence as the optimized Depth buffer clear (see above) except the **Depth Buffer Resolve Enable** bit is set. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the nest rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

Doing a resolve operation requires that a preceding Depth Buffer Clear operation is required to have initialized the Depth Clear Value.

Optimized Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation. If this occurs, the hierarchical depth

buffer must be "resolved" to avoid incorrect device behavior. This is done by setting the **Hierarchical Depth Buffer Resolve Enable** field in 3DSTATE_WM_HZ_OP and specifying a full render target sized rectangle. The depth buffer resolve uses the same sequence as the optimized Depth buffer clear (see above) except the **Hierarchical Depth Buffer Resolve Enable** bit is set. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be programmed on Xmin, Ymin, Xmax, and Ymax in the 3DSTATE_WM_HZ_OP command.
- The rectangle primitive size must be aligned to 8x4 pixels.

Separate Stencil Buffer

The following tables describe the separate stencil buffer for different generations.

The separate stencil buffer is always enabled, thus the field in 3DSTATE_DEPTH_BUFFER to explicitly enable the separate stencil buffer has been removed. Surface formats with interleaved depth and stencil are no longer supported.

The stencil buffer has a format of R8_UNIT, and shares **Surface Type**, **Height**, **Width**, and **Depth**, **Minimum Array Element**, **Render Target View Extent**, **Depth Coordinate Offset X/Y**, **LOD**, and **Depth Buffer Object Control State** fields of the depth buffer.

DepthStencil Buffer State

This section contains the state registers for the Depth/Stencil Buffers.

3DSTATE_DEPTH_BUFFER

3DSTATE_STENCIL_BUFFER

3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_CLEAR_PARAMS

Pixel Shader Thread Generation

After a group of object fragments have been rasterized, the Pixel Shader (PSD) function is invoked to further compute output information and cause results to be written to output surfaces (like color, depth, stencil, UAvs etc). Fragments can be P or S.

For each fragment, the Pixel Shader calculates the values of the various vertex attributes that are to be interpolated across the object using the interpolation coefficients. It then executes an API-supplied Pixel Shader Program. Instructions in this program permit the accessing of texture map data, where Texture Samplers are employed to sample and filter texture maps (see the Shared Functions chapter). Arithmetic operations can be performed on the texture data, input fragment information, and Pixel Shader Constants to compute the resultant fragment's output. The Pixel Shader program also allows the pixel to be discarded from further processing.

3DSTATE_PS

This command is used to set state used by the pixel shader dispatch stage.

Command
3DSTATE_PS

Programming Note	
Context:	Pixel Shader Thread Generation
Note: The PS Unit also receives 3DSTATE_PS_BLEND, 3DSTATE_SAMPLEMASK, 3DSTATE_MULTISAMPLE, and 3DSTATE_PS_EXTRA.	

Signal	PS_INT::oMask Present to RenderTarget
Description	This bit is inserted in the PS payload header and made available to the DataPort (either via the message header or via header bypass) to indicate that oMask data (one or two phases) is included in Render Target Write messages. If present, the oMask data is used to mask off samples.
Formula	= 3DSTATE_PS_EXTRA::oMask Present to RenderTarget

Signal	PS_INT::Dual Source Blend Enable
Description	This field is set if dual source blend is enabled. If this bit is disabled, the data port dual source message reverts to a single source message using source 0.
Formula	= 3DSTATE_PS_BLEND::ColorBufferBlendEnable && (PS_INT::UsesSrc1BlendFactor || (PS_INT::IndependentAlphaUsesSrc1BlendFactors && 3DSTATE_PS_BLEND::Independent Alpha Blend Enable))

Signal	PS_INT::UsesSrc1BlendFactor
Description	
Formula	= (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA)

Signal	PS_INT::IndependentAlphaUsesSrc1BlendFactors
Description	
Formula	= (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA)

Signal	PS_INT::PS UAV-only
Description	Pixel Shader UAV-only render target
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Has UAV && !3DSTATE_PS_EXTRA:: Pixel Shader Does not write to RT

Command
3DSTATE_PS_EXTRA This command is used to set state used by the pixel shader dispatch stage
3DSTATE_PS_BLEND This command is used to set state used by the pixel shader dispatch stage
3DSTATE_CONSTANT_PS

Command
3DSTATE_BINDING_TABLE_POINTERS_PS
3DSTATE_PUSH_CONSTANT_ALLOC_PS
3DSTATE_SAMPLER_STATE_POINTERS_PS

Pixel Grouping (Dispatch Size) Control

The WM unit can pass a grouping of 2 subspans (8 pixels), 4 subspans (16 pixels), or 8 subspans (32 pixels) to a Pixel Shader thread. Software should take into account the following considerations when determining which groupings to support/enable during operation. This determination involves a tradeoff of these likely conflicting issues. Note that the size of the dispatch has significant impact on the kernel program. (It is certainly not transparent to the kernel.) Also note that there is no implied spatial relationship between the subspans passed to a PS thread, other than the fact that they come from the same object.

- **Thread Efficiency**: In general, there is some amount of overhead involved with PS thread dispatch, and if this can be amortized over a larger number of pixels, efficiency will likely increase. This is especially true for very short PS kernels, as may be used for desktop composition, etc.
- **GRF Consumption**: Processing more pixels per thread requires a larger thread payload and likely more temporary register usage, both of which translate into a requirement for a larger GRF register allocation for the threads. This increased GRF usage could lead to increased use of scratch space (for spill/fill, etc.) and possibly less efficient use of the EUs (as it would be less likely to find an EU with enough free physical GRF registers to service the thread).
- **Object Size**: If the number of very small objects (e.g., covering 2 subspans or fewer) is expected to comprise a significant portion of the workload, supporting the 8-pixel dispatch mode may be advantageous. Otherwise there could be a large number of 16-pixel dispatches with only 1 or 2 valid subspans, resulting in low efficiency for those threads.
- **Intangibles**: Kernel footprint & Instruction Cache impact; Complexity; ….

The groupings of subspans that the WM unit is allowed to include in a PS thread payload is controlled by the **32,16,8 Pixel Dispatch Enable** state variables programmed in WM_STATE. Using these state variables, the WM unit attempts to dispatch the largest allowed grouping of subspans. The following table lists the possible combinations of these state variables.

Please note that, the valid column in the table indicates which products supports the combination dispatch. Combinations that are not listed in the table are not available on any product.

The letter codes A, B, D, and E used in the Variable Pixel Dispatch table below are valid for all projects with some specific mode restrictions for specific projects for B, D, and E as indicated in the next few tables.

D is like B with an added general restriction, that it cannot be used in non-1x PERSAMPLE mode.

E cannot be used in PERSAMPLE mode with number of multisamples >= 2.

Variable Pixel Dispatch

Contiguous 64 Pixel Dispatch Enable	Contiguous 32 Pixel Dispatch Enable	32 Pixel Dispatch Enable	16 Pixel Dispatch Enable	8 Pixel Dispatch Enable	Valid	IP for n-pixel Contiguous Dispatch		IP for n-pixel Dispatch (KSP offsets are in 128-bit instruction units)		
						n=64	n=32	n=32	n=16	n=8
0	0	0	0	1	A					KSP[0]
0	0	0	1	0	B				KSP[0]	
0	0	0	1	1	D				KSP[2]	KSP[0]
0	0	1	0	0	B			KSP[0]		
0	0	1	1	0	E			KSP[1]	KSP[2]	
0	0	1	1	1	D			KSP[1]	KSP[2]	KSP[0]
0	1	1	1	0	D		KSP[2]	KSP[1]	KSP[0]	
1	0	1	1	0	D	KSP[2]		KSP[1]	KSP[0]	

Each of the three KSP values is separately specified. In addition, each kernel has a separately-specified GRF register count.

Depending on the subspan grouping selected, the WM unit will modify the starting PS Instruction Pointer (derived from the Kernel Start Pointer in WM_STATE) as a means to inform the PS kernel of the number of subspans included in the payload. The modified IP is a function of the enabled modes and the dispatch size, as shown in the table below.

The driver must ensure that the PS kernel begins with a corresponding jump table to properly handle the number of subspans dispatched. The WM unit will "OR" in the two LSBs of the Kernel Pointer (bits 5:4) to create an instruction level address. (Note that the pointer from WM_STATE is 64-byte aligned which corresponds to four 128-bit instructions.)

If only one dispatch mode is enabled, the Jitter should not include any jump table entries at the beginning of the PS kernel. If multiple dispatch modes are enabled, a two entry jump table should always be inserted, regardless of which modes are enabled (jump table entry for 8 pixel dispatch, followed by jump table entry for 32 pixel dispatch).

Note that for SIMD32 dispatch, pixel shader dispatch function increments GRF Start Register for URB Data state by 2 to account for the additional SIMD16 payload. The Pixel Shader kernel needs to comprehend this modification for SIMD32.

```
if ( 32PixelDispatchEnable && n > 7 )
    Dispatch 32 Pixels
else if ( 16PixelDispatchEnable && ( n > 2 || ! 8PixelDispatchEnable) )
    Dispatch 16 Pixels
else
    Dispatch 8 Pixels
```

```
end if
```

Multisampling Effects on Pixel Shader Dispatch

The pixel shader payloads are defined in terms of subspans and pixels. The slots in the pixel shader thread previously mapped 1:1 with pixels. With multisampling, a slot could contain a pixel or may just contain a single sample, depending on the mode. Payload definitions now refer to *slot* to make the definition independent of multisampling mode.

MSDISPMODE_PERPIXEL Thread Dispatch

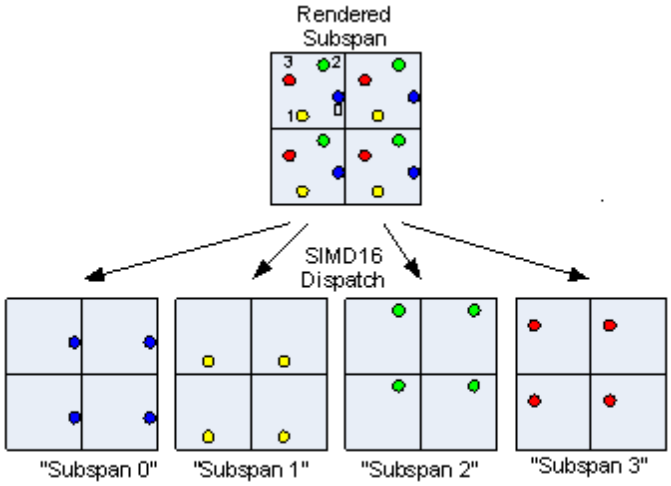
In PERPIXEL mode, the pixel shader kernel still works on 2/4/8 separate subspans, depending on dispatch mode. The fact that rasterization and the depth/stencil tests are being performed on a per-sample (not per-pixel) basis is transparent to the pixel shader kernel.

Programming Note	
Context:	MSDISPMODE_PERPIXEL Thread Dispatch
When NUM_MULTISAMPLES == 16 (i.e. 16x MSAA) and PS_DISPATCH_MODE is PER_PIXEL, SIMD32 pixel shader is not supported.	

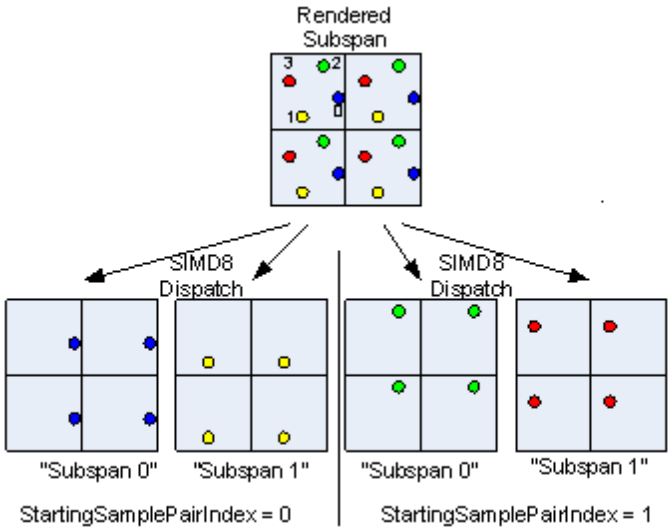
MSDISPMODE_PERSAMPLE Thread Dispatch

In PERSAMPLE mode, the pixel shader needs to operate on a sample vs. pixel basis (although this collapses in NUMSAMPLES_1 mode) Instead of processing strictly different subspans in parallel , the PS kernel processes different sample indices of one or more subspans in parallel For example, a SIMD16 dispatch in PERSAMPLE/NUMSAMPLES_4 mode would operate on a single subspan, with the usual "4 Subspan0 pixel slots" used for the "4 Sample0 locations of the (single) subspan" Subspan1 slots would be used for the Sample1 locations, and so on This layout allows the pixel shader to compute derivatives/LOD based on deltas between corresponding sample locations in the subspan in the same fashion as LEGACY pixel shader execution, and as required by DX10.1.

Depending on the dispatch mode (8/16/32 pixels) and multisampling mode (1X/4X), there are different mappings of subspans/samples onto dispatches and slots-within-dispatch In some cases, more than one subspan may be included in a dispatch, while in other cases multiple dispatches are be required to process all samples for a single subspan In the latter case, the **StartingSamplePairIndex** value is included in the payload header so the Render Target Write message will access the correct samples with each message.



PERSAMPLE SIMD16 4X Dispatch



PERSAMPLE Dispatch

The following table provides the complete dispatch/slot mappings for all the MS/Dispatch combinations.

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]*

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0] *Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0]* *Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0]* *Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]* *Slot[19:16] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[23:20] = Subspan[2].Pixel[3:0].Sample[1]* *Slot[27:24] = Subspan[3].Pixel[3:0].Sample[0]* *Slot[31:28] = Subspan[3].Pixel[3:0].Sample[1]*
	4X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]* *Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1]* *Slot[27:24] = Subspan[1].Pixel[3:0].Sample[2]* *Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]*
	8X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]* *Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4]* *Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5]* *Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6]* *Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]*
	16x	*Dispatch[i]: (i=0, 4)*

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		SSPI = i *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]* *Slot[19:16] = Subspan[0].Pixel[3:0].Sample[SSPI*2+4]* *Slot[23:20] = Subspan[0].Pixel[3:0].Sample[SSPI*2+5]* *Slot[27:24] = Subspan[0].Pixel[3:0].Sample[SSPI*2+6]* *Slot[31:28] = Subspan[0].Pixel[3:0].Sample[SSPI*2+7]*
SIMD16	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]*
	4X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]*
	8X	*Dispatch[i]: (i=0, 2)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]*
	16x	*Dispatch[i]: (i=0, 2, 4, 6)*

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		SSPI = i *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]*
SIMD8	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]*
	4X	*Dispatch[i]: (i=0..1)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]*
	8X	*Dispatch[i]: (i=0, 1, 2, 3)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]*
	16x	*Dispatch[i]: (i=0, 1, 2, 3, 4, 5, 6, 7)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]*

PS Thread Payload for Normal Dispatch

The following table lists all possible contents included in a PS thread payload, in the order they are provided. Certain portions of the payload are optional, in which case the corresponding phase is skipped.

This payload does not apply to the contiguous dispatch modes. The payload for these modes is documented in the section titled *PS Thread Payload for Contiguous Dispatch*.

PS Thread Payload for Normal Dispatch

All registers are numbered starting at 0, but many registers are skipped depending on configuration. This causes all registers below to be renumbered to fill in the skipped locations. The only case where actual registers may be skipped is immediately before the constant data and again before the setup data.

PS Thread Payload for Normal Dispatch

DWord	Bits	Description
R0.7	31	Reserved.
	30:24	Reserved
	23:0	**Primitive Thread ID:** This field contains the primitive thread count passed to the Windower from the Strips Fans Unit. Format: Reserved for HW Implementation Use.
R0.6	31:24	Reserved
	23:0	**Thread ID:** This field contains the thread count which is incremented by the Windower for every thread that is dispatched. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer:** Specifies the 1K-byte aligned pointer to the scratch space available for this PS thread. This is specified as an offset to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Reserved
	7:0	**FFTID:** This ID is assigned by the WM unit and is an identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the Sampler State table. It is specified as an offset from the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>thread.

Programming Notes: This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.

Format = U4

Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two</td></tr>
<tr><td>R0.2</td><td>31:0</td><td>Reserved: Delivered as zeros (reserved for message header fields).</td></tr>
<tr><td rowspan="2">R0.1</td><td>31:6</td><td>Color Calculator State Pointer: Specifies the 64-byte aligned pointer to the Color Calculator state (COLOR_CALC_STATE structure in memory). It is specified as an offset from the Dynamic State Base Address. This value is eventually passed to the ColorCalc function in the DataPort and is used to fetch the corresponding CC_STATE data.

Format = DynamicStateOffset[31:5]</td></tr>
<tr><td>5:0</td><td>Reserved</td></tr>
<tr><td rowspan="4">R0.0</td><td>31</td><td>Reserved</td></tr>
<tr><td>30:27</td><td>Viewport Index: Specifies the index of the viewport currently being used.

Format = U4

Range = [0,15]</td></tr>
<tr><td>26:16</td><td>Render Target Array Index: Specifies the array index to be used for the following surface types:

SURFTYPE_1D: specifies the array index Range = [0,2047]

SURFTYPE_2D: specifies the array index Range = [0,2047]

SURFTYPE_3D: specifies the “r” coordinate Range = [0,2047]

SURFTYPE_CUBE: specifies the face identifier Range = [0,5]

<table><tr><th>Face</th><th>Render Target Array Index</th></tr><tr><td>+x</td><td>0</td></tr><tr><td>-x</td><td>1</td></tr><tr><td>+y</td><td>2</td></tr><tr><td>-y</td><td>3</td></tr><tr><td>+z</td><td>4</td></tr><tr><td>-z</td><td>5</td></tr></table>
Format = U11</td></tr>
<tr><td>15</td><td>Front/Back Facing Polygon: Determines whether the polygon is front or back facing. Used by</td></tr>
</table>

DWord	Bits	Description
		the render cache to determine which stencil test state to use. 0: Front Facing 1: Back Facing
	14	Reserved
	13	**Source Depth to Render Target:** Indicates that source depth will be sent to the render target.
	12	**oMask to Render Target:** Indicates that oMask will be sent to the render target.
	11:9	Reserved
	8	Reserved for expansion of **Starting Sample Pair Index**.
	7:6	**Starting Sample Pair Index:** Indicates the index of the first sample pair of the dispatch. Format = U2 Range = [0,3]
	5	Reserved
	4:0	**Primitive Topology Type:** This field identifies the Primitive Topology Type associated with the primitive spawning this object. The WM unit does not modify this value (e.g., objects within POINTLIST topologies see POINTLIST). Format: (See 3DPRIMITIVE command in *3D Pipeline*.)
R1.7	31:16	**Pixel/Sample Mask (SubSpan[3:0]):** Indicates which pixels within the four subspans are lit. If 32 pixel dispatch is enabled, this field contains the pixel mask for the first four subspans. **Note:** This is not a duplicate of the Dispatch Mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[3:0]):** This is a duplicate copy of the pixel mask. This copy can be modified as the pixel shader thread executes in order to turn off pixels based on kill instructions.
R1.6	31:0	Reserved
R1.5	31:16	**Y3:** Y coordinate (screen space) for upper-left pixel of subspan 3 (slot 12). Format = U16
	15:0	**X3:** X coordinate (screen space) for upper-left pixel of subspan 3 (slot 12). Format = U16

DWord	Bits	Description
R1.4	31:16	**Y2:** Y coordinate (screen space) for upper-left pixel of subspan 2 (slot 8). Format = U16
	15:0	**X2:** X coordinate (screen space) for upper-left pixel of subspan 2 (slot 8). Format = U16
R1.3	31:16	**Y1:** Y coordinate (screen space) for upper-left pixel of subspan 1 (slot 4). Format = U16
	15:0	**X1:** X coordinate (screen space) for upper-left pixel of subspan 1 (slot 4). Format = U16
R1.2	31:16	**Y0:** Y coordinate (screen space) for upper-left pixel of subspan 0 (slot 0). Format = U16
	15:0	**X0:** X coordinate (screen space) for upper-left pixel of subspan 0 (slot 0). Format = U16
R1.0	31:20	Reserved
	15:12	Slot 3 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	11:8	Slot 2 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	7:4	Slot 1 SampleID (if pixel or sample dispatch) Format = U4

DWord	Bits	Description
		1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	3:0	Slot 0 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
		R2: Delivered only if this is a *32-pixel dispatch*.
R2.7	31:16	**Pixel/Sample Mask (SubSpan[7:4]):** Indicates which pixels within the upper four subspans are lit. This field is valid only when the 32 pixel dispatch state is enabled. This field must not be modified by the pixel shader thread. Note: This is not a duplicate of the dispatch mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[7:4]):** This is a duplicate copy of pixel mask for the upper 16 pixels. This copy will be modified as the pixel shader thread executes to turn off pixels based on kill instructions.
R2.6	31:0	Reserved
R2.5	31:16	**Y7:** Y coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
	15:0	**X7:** X coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
R2.4	31:16	Y6
	15:0	X6
R2.3	31:16	Y5
	15:0	X5
R2.2	31:16	Y4

DWord	Bits	Description
	15:0	X4
R2.1	31:0	Reserved
R2.0	31:16	Reserved
	15:12	Slot 7 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	11:8	Slot 6 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	7:4	Slot 5 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	3:0	Slot 4 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]

DWord	Bits	Description
		16X MSAA range [0..15]
		R3-R26: Delivered only if the corresponding **Barycentric Interpolation Mode** bit is set. Register phases containing Slot 8-15 data are not delivered in *8-pixel dispatch* mode.
R3.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 7 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R3.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 6
R3.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 5
R3.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 4
R3.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 3
R3.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 2
R3.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 1
R3.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 0
R4		Perspective Pixel Location Barycentric[2] for Slots 7:0
R5.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 15
R5.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 14
R5.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 13
R5.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 12
R5.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 11
R5.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 10
R5.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 9
R5.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 8
R6		Perspective Pixel Location Barycentric[2] for Slots 15:8
R7:10		Perspective Centroid Barycentric
R11:14		Perspective Sample Barycentric
R15:18		Linear Pixel Location Barycentric
R19:22		Linear Centroid Barycentric
R23:26		Linear Sample Barycentric
		R27: Delivered only if **Pixel Shader Uses Source Depth** is set.
R27.7	31:0	Interpolated Depth for Slot 7 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth**

DWord	Bits	Description
		(WM_STATE) is set.
R27.6	31:0	Interpolated Depth for Slot 6
R27.5	31:0	Interpolated Depth for Slot 5
R27.4	31:0	Interpolated Depth for Slot 4
R27.3	31:0	Interpolated Depth for Slot 3
R27.2	31:0	Interpolated Depth for Slot 2
R27.1	31:0	Interpolated Depth for Slot 1
R27.0	31:0	Interpolated Depth for Slot 0
		R28: Delivered only if **Pixel Shader Uses Source Depth** is set and this is not an *8-pixel dispatch*.
R28.7	31:0	Interpolated Depth for Slot 15
R28.6	31:0	Interpolated Depth for Slot 14
R28.5	31:0	Interpolated Depth for Slot 13
R28.4	31:0	Interpolated Depth for Slot 12
R28.3	31:0	Interpolated Depth for Slot 11
R28.2	31:0	Interpolated Depth for Slot 10
R28.1	31:0	Interpolated Depth for Slot 9
R28.0	31:0	Interpolated Depth for Slot 8
		R29: Delivered only if **Pixel Shader Uses Source W** is set.
R29.7	31:0	Interpolated W for Slot 7 Format = IEEE_Float This and the next register phase are only included if **Pixel Shader Uses Source W** (WM_STATE) is set.
R29.6	31:0	Interpolated W for Slot 6
R29.5	31:0	Interpolated W for Slot 5
R29.4	31:0	Interpolated W for Slot 4
R29.3	31:0	Interpolated W for Slot 3
R29.2	31:0	Interpolated W for Slot 2
R29.1	31:0	Interpolated W for Slot 1
R29.0	31:0	Interpolated W for Slot 0
		R30: Delivered only if **Pixel Shader Uses Source W** is set and this is not an *8-pixel dispatch*.
R30.7	31:0	Interpolated W for Slot 15
R30.6	31:0	Interpolated W for Slot 14
R30.5	31:0	Interpolated W for Slot 13

DWord	Bits	Description
R30.4	31:0	Interpolated W for Slot 12
R30.3	31:0	Interpolated W for Slot 11
R30.2	31:0	Interpolated W for Slot 10
R30.1	31:0	Interpolated W for Slot 9
R30.0	31:0	Interpolated W for Slot 8
		R31: Delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE.
R31.7	31:24	Position Offset Y for Slot 15 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 For non-CP rate dispatch: Range = [0.0,1.0) For CP rate dispatch: Range = [0.0,4.0)
	23:16	Position Offset X for Slot 15 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of **Position XY Offset Select**. Format = U4.4 For non-CP rate dispatch: Range = [0.0,1.0) For CP rate dispatch: Range = [0.0,4.0)
	15:8	Position Offset Y for Slot 14
	7:0	Position Offset X for Slot 14
R31.6	31:24	Position Offset Y for Slot 13
	23:16	Position Offset X for Slot 13
	15:8	Position Offset Y for Slot 12
	7:0	Position Offset X for Slot 12
R31.5:4		Position Offset X/Y for Slot[11:8]
R31.3:2		Position Offset X/Y for Slot[7:4]
R31.1:0		Position Offset X/Y for Slot[3:0]
		R32: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set.
R32.7	31:0	Input Coverage Mask for Slot 7 Format = U32 This and the next register phase is only included if **Pixel Shader Uses Input Coverage Mask**

DWord	Bits	Description
		(3DSTATE_PS) is set. This field always encodes sample Coverage Mask.
R32.6	31:0	Input Coverage Mask for Slot 6
R32.5	31:0	Input Coverage Mask for Slot 5
R32.4	31:0	Input Coverage Mask for Slot 4
R32.3	31:0	Input Coverage Mask for Slot 3
R32.2	31:0	Input Coverage Mask for Slot 2
R32.1	31:0	Input Coverage Mask for Slot 1
R32.0	31:0	Input Coverage Mask for Slot 0
		R33: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set and this is not an *8-pixel dispatch*.
R33.7	31:0	Input Coverage Mask for Slot 15
R33.6	31:0	Input Coverage Mask for Slot 14
R33.5	31:0	Input Coverage Mask for Slot 13
R33.4	31:0	Input Coverage Mask for Slot 12
R33.3	31:0	Input Coverage Mask for Slot 11
R33.2	31:0	Input Coverage Mask for Slot 10
R33.1	31:0	Input Coverage Mask for Slot 9
R33.0	31:0	Input Coverage Mask for Slot 8
		R34-R57: Delivered only if the corresponding **Barycentric Interpolation Mode** bit is set and this is a *32-pixel dispatch*.
R34.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 23 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R34.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 22
R34.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 21
R34.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 20
R34.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 19
R34.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 18
R34.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 17
R34.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 16
R35		Perspective Pixel Location Barycentric[2] for Slots 23:16
R36.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 31

DWord	Bits	Description
R36.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 30
R36.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 29
R36.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 28
R36.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 27
R36.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 26
R36.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 25
R36.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 24
R37		Perspective Pixel Location Barycentric[2] for Slots 31:24
R38:41		Perspective Centroid Barycentric
R42:45		Perspective Sample Barycentric
R46:49		Linear Pixel Location Barycentric
R50:53		Linear Centroid Barycentric
R54:57		Linear Sample Barycentric
		R58-R59: Delivered only if **Pixel Shader Uses Source Depth** is set and this is a *32-pixel dispatch*.
R58.7	31:0	Interpolated Depth for Slot 23 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth** (WM_STATE) bit is set.
R58.6	31:0	Interpolated Depth for Slot 22
R58.5	31:0	Interpolated Depth for Slot 21
R58.4	31:0	Interpolated Depth for Slot 20
R58.3	31:0	Interpolated Depth for Slot 19
R58.2	31:0	Interpolated Depth for Slot 18
R58.1	31:0	Interpolated Depth for Slot 17
R58.0	31:0	Interpolated Depth for Slot 16
R59.7	31:0	Interpolated Depth for Slot 31
R59.6	31:0	Interpolated Depth for Slot 30
R59.5	31:0	Interpolated Depth for Slot 29
R59.4	31:0	Interpolated Depth for Slot 28
R59.3	31:0	Interpolated Depth for Slot 27
R59.2	31:0	Interpolated Depth for Slot 26
R59.1	31:0	Interpolated Depth for Slot 25
R59.0	31:0	Interpolated Depth for Slot 24
		R60-R61: Delivered only if **Pixel Shader Uses Source W** is set and this is a *32-pixel dispatch*.

DWord	Bits	Description
R60.7	31:0	Interpolated W for Slot 23 Format = IEEE_Float This and the next register phase are only included if **Pixel Shader Uses Source W** (WM_STATE) bit is set.
R60.6	31:0	Interpolated W for Slot 22
R60.5	31:0	Interpolated W for Slot 21
R60.4	31:0	Interpolated W for Slot 20
R60.3	31:0	Interpolated W for Slot 19
R60.2	31:0	Interpolated W for Slot 18
R60.1	31:0	Interpolated W for Slot 17
R60.0	31:0	Interpolated W for Slot 16
R61.7	31:0	Interpolated W for Slot 31
R61.6	31:0	Interpolated W for Slot 30
R61.5	31:0	Interpolated W for Slot 29
R61.4	31:0	Interpolated W for Slot 28
R61.3	31:0	Interpolated W for Slot 27
R61.2	31:0	Interpolated W for Slot 26
R61.1	31:0	Interpolated W for Slot 25
R61.0	31:0	Interpolated W for Slot 24
		R62: Delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE and this is a *32-pixel dispatch*.
R62.7	31:24	Position Offset Y for Slot 31 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 31 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 30
	7:0	Position Offset X for Slot 30

DWord	Bits	Description
R62.6	31:24	Position Offset Y for Slot 29
	23:16	Position Offset X for Slot 29
	15:8	Position Offset Y for Slot 28
	7:0	Position Offset X for Slot 28
R62.5:4		Position Offset X/Y for Slot[27:24]
R62.3:2		Position Offset X/Y for Slot[23:20]
R62.1:0		Position Offset X/Y for Slot[19:16]
		R63-R64: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set and this is a *32-pixel dispatch*.
R63.7	31:0	Input Coverage Mask for Slot 23 Format = U32 This and the next register phase are only included if **Pixel Shader Uses Input Coverage Mask** (3DSTATE_PS) is set.
R63.6	31:0	Input Coverage Mask for Slot 22
R63.5	31:0	Input Coverage Mask for Slot 21
R63.4	31:0	Input Coverage Mask for Slot 20
R63.3	31:0	Input Coverage Mask for Slot 19
R63.2	31:0	Input Coverage Mask for Slot 18
R63.1	31:0	Input Coverage Mask for Slot 17
R63.0	31:0	Input Coverage Mask for Slot 16
R64.7	31:0	Input Coverage Mask for Slot 31
R64.6	31:0	Input Coverage Mask for Slot 30
R64.5	31:0	Input Coverage Mask for Slot 29
R64.4	31:0	Input Coverage Mask for Slot 28
R64.3	31:0	Input Coverage Mask for Slot 27
R64.2	31:0	Input Coverage Mask for Slot 26
R64.1	31:0	Input Coverage Mask for Slot 25
R64.0	31:0	Input Coverage Mask for Slot 24
[Varies] optional	255:0	**Constant Data (optional):** For more details about the size and source of constant data, please refer to General Programming of Thread-Generating Stages in the Push Constants chapter.

Pixel Backend

This section contains the following subsections:

- MCS Buffer for Render Target(s)
- Render Target Fast Clear
- Render TargetResolve

Color Calculator (Output Merger)

Overview

Note: The Color Calculator logic resides in the Render Cache backing Data Port (DAP) shared function. It is described in this chapter as the Color Calc functions are naturally an extension of the 3D pipeline past the WM stage. See the DataPort chapter for details on the messages used by the Pixel Shader to invoke Color Calculator functionality.

The *Color Calculator* (referred to as "Output Merger in the DX Spec) function within the Data Port shared function completes the processing of rasterized pixels after the pixel color and depth have been computed by the Pixel Shader. This processing is initiated when the pixel shader thread sends a Render Target Write message (see *Shared Functions*) to the Render Cache. (Note that a single pixel shader thread may send multiple Render Target Write messages, with the result that multiple render targets get updated.) The pixel variables pass through a pipeline of fixed (yet programmable) functions, and the results are conditionally written into the appropriate buffers.

The word "pixel" used in this section is effectively replaced with the word "sample" if multisample rasterization is enabled.

Pipeline Stage	Description
Alpha Coverage	It generates coverage masks using AlphaToCoverage AND/OR AlphaToOne functions based on src0.alpha.
Alpha Test	Compare pixel alpha with reference alpha and conditionally discard pixel.
Stencil Test	Compare pixel stencil value with reference and forward result to Buffer Update stage.
Depth Test	Compare pix.Z with corresponding Z value in the Depth Buffer and forward result to Buffer Update stage.
Color Blending	Combine pixel color with corresponding color in color buffer according to programmable function.
Gamma Correction	Adjust pixel's color according to gamma function for SRGB destination surfaces.
Color Quantization	Convert "full precision" pixel color values to fixed precision of the color buffer format.
Logic Ops	Combine pixel color logically with existing color buffer color (mutually exclusive with Color Blending).
Buffer Update	Write final pixel values to color and depth buffers or discard pixel without update.

The following logic describes the high-level operation of the Pixel Processing pipeline:

```
PixelProcessing() {
    AlphaCoverage()
    AlphaTest()
    DepthBufferCoordinateOffsetDisable
    StencilTest()
    DepthTest()
    ColorBufferBlending()
    GammaCorrection()
    ColorQuantization()
    LogicalOps()
    BufferUpdate()
}
```

Alpha Coverage

Alpha coverage logic is supported for SNB+ and can be controlled using three state variables:

- **AlphaToCoverage Enable**, when enabled Color Calculator modifies the sample mask. This function (along with AlphaToOne) come at the top of the pixel pipeline. The sample's Source0.Alpha value (possibly being replicated from the pixel's Source0.Alpha) is used to compute a (optionally dithered) 1/2/4-bit mask (depending on NumSamples).
- The **AlphaToCoverage Dither Enable** SV is used to control the dithering of the AlphaToCoverage mask. The bit corresponding to the sample# is then ANDed with the sample's incoming mask bits – allowing the sample to be masked off depending on alpha.
- **AlphaToOne Enable,** when enabled, Color Calculator must replace Source0.Alpha (if present) with 1.0f.
- If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.

If Pixel Shader outputs oMask, AlphaToCoverage is disabled in hardware, regardless of the state setting for this feature.

Notes:

- Src0.alpha needs to be first multiplied with AA alpha before applying AlphaToCoverage and AlphaToOne functions.
- An alpha value of NaN results in a no coverage (zero) mask.
- Alpha values from the pixel shader are treated as FLOAT32 format for computing the AlphaToCoverage Mask.

Alpha Test

The Alpha Test function can be used to discard pixels based on a comparison between the incoming pixel's alpha value and the **Alpha Test Reference** state variable in COLOR_CALC_STATE. This operation can be used to remove transparent or nearly-transparent pixels, though other uses for the alpha channel and alpha test are certainly possible.

This function is enabled by the **Alpha Test Enable** state variable in COLOR_CALC_STATE. If ENABLED, this function compares the incoming pixel's alpha value (*pixColor.Alpha*) and the reference alpha value specified by via the **Alpha Test Reference** state variable in COLOR_CALC_STATE. The comparison performed is specified by the **Alpha Test Function** state variable in COLOR_CALC_STATE.

The **Alpha Test Format** state variable is used to specify whether Alpha Test is performed using fixed-point (UNORM8) or FLOAT32 values. Accordingly, it determines whether the **Alpha Reference Value** is passed in a UNORM8 or FLOAT32 format. If UNORM8 is selected, the pixel's alpha value will be converted from floating-point to UNORM8 before the comparison.

Pixels that pass the Alpha Test proceed for further processing. Those that fail are discarded at this point in the pipeline.

If **Alpha Test Enable** is DISABLED, this pipeline stage has no effect.

The Alpha Test function is supported in conjunction with Multiple Render Targets (MRTs). If delivered in the incoming render target write message, source 0 alpha is used to perform the alpha test. If source 0 alpha is not delivered, the normal alpha value is used to perform the alpha test.

Depth Coordinate Offset

The Depth Coordinate Offset function applies a programmable constant offset to the RenderTarget X,Y screen space coordinates in order to generate DepthBuffer coordinates.

The function has been specifically added to allow the OpenGL driver to deal with a RenderTarget and DepthBuffer of differing sizes.

This condition is not an issue for the D3D driver, as D3D defines an upper-left screen coordinate origin which matches the HW rasterizer; as long as the application limits rendering to the smaller of the RT/DepthBuffer extents, no special logic is required.

OpenGL defines a lower-left screen coordinate origin. This requires the driver to incorporate a "Y coordinate flipping" transformation into the viewport mapping function. The Y extent of the RT is used in this flipping transformation. If the DepthBuffer extent is different, the wrong pixel Y locations within the DepthBuffer will be accessed.

The least expensive solution is to provide a translation offset to be applied to the post-viewport-mapped DepthBuffer Y pixel coordinate, effectively allowing the alignment of the lower-left origins of the RT and DepthBuffer. [Note that the previous DBCOD feature performed an optional translation of post-viewport-mapping RT pixel (screen) coordinates to generate DepthBuffer pixel (window) coordinates. Specifically, the Draw Rect Origin X,Y state could be subtracted from the RT pixel coordinates.]

This function uses **Depth Coordinate Offset X,Y** state (signed 16-bit values in 3DSTATE_DEPTH_RECTANGLE) that is unconditionally added to the RT pixel coordinates to generate DepthBuffer pixel coordinates.

The previous DBCOB feature can be supported by having the driver program Depth Coordinate X,Y Offset to the two's complement of the the Draw Rect Origin. By programming Depth Coordinate X,Y Offset to zeros, the current "normal" operation (DBCOD disabled) can be achieved.

Programming Note	
Context:	Depth Coordinate Offset
• Only simple 2D RTs are supported (no mipmaps). • Software must ensure that the resultant DepthBuffer Coordinate X,Y values are non-negative. • There are alignment restrictions – see 3DSTATE_DEPTH_BUFFER command.on SFID_DP_DC2) are IA coherent.	

Stencil Test

The Stencil Test function can be used to discard pixels based on a comparison between the [**Backface**] **Stencil Test Reference** state variable and the pixel's stencil value. This is a general purpose function used for such effects as shadow volumes, per-pixel clipping, etc. The result of this comparison is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Stencil Test Enable** state variable. If ENABLED, the current stencil buffer value for this pixel is read.

A 2nd set of the stencil test state variables is provided so that pixels from back-facing objects, assuming they are not culled, can have a stencil test performed on them separate from the test for normal front-facing objects. The separate stencil test for back-facing objects can be enabled via the **Double Sided Stencil Enable** state variable. Otherwise, non-culled back-facing objects will use the same test function, mask and reference value as front-facing objects. The 2nd stencil state for back-facing objects is most commonly used to improve the performance of rendering shadow volumes which require a different stencil buffer operation depending on whether pixels rendered are from a front-facing or back-facing object. The backface stencil state removes the requirement to render the shadow volumes in 2 passes or sort the objects into front-facing and back-facing lists.

The remainder of this subsection describes the function in term of [**Backface**] <**state variable name**>. The Backface set of state variables are only used if Double Sided Stencil Enable is ENABLED and the object is considered back-facing. Otherwise the normal (front-facing) state variables are used.

This function then compares the [**Backface**] **Stencil Test Reference** value and the pixel's stencil value value after logically ANDing both values by [**Backface**] **Stencil Test Mask**. The comparison performed is specified by the [**Backface**] **Stencil Test Function** state variable. The result of the comparison is passed down the pipeline for use in the Stencil Buffer Update function. The Stencil Test function does not in itself discard pixels.

If **Stencil Test Enable** is DISABLED, a result of "stencil test passed" is propagated down the pipeline.

Depth Test

The Depth Test function can be used to discard pixels based on a comparison between the incoming pixel's depth value and the current depth buffer value associated with the pixel. This function is typically used to perform the "Z Buffer" hidden surface removal. The result of this pipeline function is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Depth Test Enable** state variable. If enabled, the pixel's ("source") depth value is first computed. After computation the pixel's depth value is clamped to the range defined by **Minimum Depth** and **Maximum Depth** in the selected CC_VIEWPORT state. Then the current ("destination") depth buffer value for this pixel is read.

This function then compares the source and destination depth values. The comparison performed is specified by the **Depth Test Function** state variable.

The result of the comparison is propogated down the pipeline for use in the subsequent Depth Buffer Update function. The Depth Test function does not in itself discard pixels.

If **Depth Test Enable** is DISABLED, a result of "depth test passed" is propagated down the pipeline.

Programming Note:

- Enabling the Depth Test function without defining a Depth Buffer is UNDEFINED.

Pre-Blend Color Clamping

Pre-Blend Color Clamping, controlled via **Pre-Blend Color Clamp Enable** OR **Pre-Blend Source Only Clamp Enable** and **Color Clamp Range** states in COLOR_CALC_STATE, is affected by the enabling of Color Buffer Blend as described below.

The following table summarizes the requirements involved with Pre-/Post-Blend Color Clamping.

Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
Off	UNORM, UNORM_SRGB,YCRCB	Must be enabled with range = RT range or [0,1] (same function)	N/A, state ignored
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	N/A, state ignored
	FLOAT (except for R11G11B10_FLOAT)	Must be enabled (with any desired range)	N/A, state ignored
	R11G11B10_FLOAT	Must be enabled with either [0,1] or RT range	N/A, state ignored
	UINT, SINT	State ignored, implied clamp to RT range	N/A, state ignored
On (where permitted)	UNORM, UNORM_SRGB	Must be enabled with range = RT range or [0,1] (same function)	Must be enabled with range = RT range or [0,1] (same function)
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	Must be enabled with range = RT range or [-1,1] (same function)
	FLOAT (except for R11G11B10_FLOAT)	Can be disabled or enabled (with any desired range)	Must be enabled (with any desired range)
	R11G11B10_FLOAT	Can be disabled or enabled (with any desired range)	Must be enabled with either [0,1] or RT range

Pre-Blend Color Clamping When Blending is Disabled

The clamping of source color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all source color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed.

Programming Note	
Context:	Pre-Blend Color Clamping When Blending is Disabled

- Given the possibility of writing UNPREDICTABLE values to the Color Buffer, it is expected and highly recommended that, when blending is disabled, software set **Pre-Blend Color Clamp Enable** to ENABLED and select an appropriate **Color Clamp Range**.
- When using SINT or UINT rendertarget surface formats, **Blending must** be DISABLED. The **Pre-Blend Color Clamp Enable** and **Color Clamp Range** fields are ignored, and an implied clamp to the rendertarget surface format is performed.

Pre-Blend Color Clamping When Blending is Enabled

The clamping of source, destination and constant color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all these color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed on these color components prior to blending.

Color Buffer Blending

The Color Buffer Blending function is used to combine one or two incoming "source" pixel color+alpha values with the "destination" color+alpha read from the corresponding location in a RenderTarget.

Blending is enabled on a global basis by the **Color Buffer Blend Enable** state variable (in COLOR_CALC_STATE). If DISABLED, Blending and Post-Blend Clamp functions are disabled for all RenderTargets, and the pixel values (possibly subject to Pre-Blend Clamp) are passed through unchanged.

The Color Buffer Blend Enable is in the per-render-target BLEND_STATE, and the field in SURFACE_STATE is no longer supported.

Programming Note	
Context:	Color Buffer Blending, Logic Ops, DataPort, surface formats, render targets

- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Dual source blending: The DataPort only supports dual source blending with a SIMD8-style message.
- Only certain surface formats support Color Buffer Blending. Refer to the Surface Format tables in *Sampling Engine*. Blending must be disabled on a RenderTarget if blending is not supported.

The incoming "source" pixel values are modulated by a selected "source" blend factor, and the possibly gamma-decorrected "destination" values are modulated by a "destination" blend factor. These terms are then combined with a "blend function". In general:

src_term = src_blend_factor * src_color

dst_term = dst_blend_factor * dst_color

color output = blend_function(src_term, dst_term)

If there is no alpha value contained in the Color Buffer, a default value of 1.0 is used and, correspondingly, there is no alpha component computed by this function.

Dual Source Blending: When using "Dual Source" Render Target Write messages, the Source1 pixel color+alpha passed in the message can be selected as a src/dst blend factor. See Color Buffer Blend Color Factors. In single-source mode, those blend factor selections are invalid. If SRC1 is included in a src/dst blend factor and a DualSource RT Write message is not used, results are UNDEFINED. (This reflects the same restriction in DX APIs, where undefined results are produced if "o1" is not written by a PS – there are no default values defined). If SRC1 is not included in a src/dst blend factor, dual source blending must be disabled.

The blending of the color and alpha components is controlled with two separate (color and alpha) sets of state variables. However, if the **Independent Alpha Blend Enable** state variable in COLOR_CALC_STATE is DISABLED, then the "color" (rather than "alpha") set of state variables is used for both color and alpha. Note that this is the only use of the **Independent Alpha Blend Enable** state – it does not control whether Blending occurs, only how.

Per **Render Target Blend State**: Blend state is selected based on **Render Target Index** contained in the message header, and appropriate blend state is applied to Render Target Write messages.

The following table describes the color source and destination blend factors controlled by the **Source [Alpha] Blend Factor** and **Destination [Alpha] Blend Factor** state variables in COLOR_CALC_STATE. Note that the blend factors applied to the R,G,B channels are always controlled by the **Source/Destination Blend Factor**, while the blend factor applied to the alpha channel is controlled either by **Source/Destination Blend Factor** or **Source/Destination Alpha Blend Factor**.

Color Buffer Blend Color Factors

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2nd output from PS in Dual-Souce mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_ZERO	0.0, 0.0, 0.0, 0.0
BLENDFACTOR_ONE	1.0, 1.0, 1.0, 1.0
BLENDFACTOR_SRC_COLOR	oN.r, oN.g, oN.b, oN.a
BLENDFACTOR_INV_SRC_COLOR	1.0-oN.r, 1.0-oN.g, 1.0-oN.b, 1.0-oN.a
BLENDFACTOR_SRC_ALPHA	oN.a, oN.a, oN.a, oN.a
BLENDFACTOR_INV_SRC_ALPHA	1.0-oN.a, 1.0-oN.a, 1.0-oN.a, 1.0-oN.a
BLENDFACTOR_SRC1_COLOR	o1.r, o1.g, o1.b, o1.a

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2nd output from PS in Dual-Souce mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_INV_SRC1_COLOR	1.0-o1.r, 1.0-o1.g, 1.0-o1.b, 1.0-o1.a
BLENDFACTOR_SRC1_ALPHA	o1.a, o1.a, o1.a, o1.a
BLENDFACTOR_INV_SRC1_ALPHA	1.0-o1.a, 1.0-o1.a, 1.0-o1.a, 1.0-o1.a
BLENDFACTOR_DST_COLOR	rtN.r, rtN.g, rtN.b, rtN.a
BLENDFACTOR_INV_DST_COLOR	1.0-rtN.r, 1.0-rtN.g, 1.0-rtN.b, 1.0-rtN.a
BLENDFACTOR_DST_ALPHA	rtN.a, rtN.a, rtN.a, rtN.a
BLENDFACTOR_INV_DST_ALPHA	1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a
BLENDFACTOR_CONST_COLOR	CC.r, CC.g, CC.b, CC.a
BLENDFACTOR_INV_CONST_COLOR	1.0-CC.r, 1.0-CC.g, 1.0-CC.b, 1.0-CC.a
BLENDFACTOR_CONST_ALPHA	CC.a, CC.a, CC.a, CC.a
BLENDFACTOR_INV_CONST_ALPHA	1.0-CC.a, 1.0-CC.a, 1.0-CC.a, 1.0-CC.a
BLENDFACTOR_SRC_ALPHA_SATURATE	f,f,f,1.0 where f = min(1.0 – rtN.a, oN.a)

The following table lists the supported blending operations defined by the **Color Blend Function** state variable and the **Alpha Blend Function** state variable (when in independent alpha blend mode).

Color Buffer Blend Functions

Blend Function	Operation (for each color component)
BLENDFUNCTION_ADD	SrcColor*SrcFactor + DstColor*DstFactor
BLENDFUNCTION_SUBTRACT	SrcColor*SrcFactor - DstColor*DstFactor
BLENDFUNCTION_REVERSE_SUBTRACT	DstColor*DstFactor - SrcColor*SrcFactor
BLENDFUNCTION_MIN	min (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "min" function.
BLENDFUNCTION_MAX	max (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "max" function.

Post-Blend Color Clamping

(See *Pre-Blend Color Clamping* above for a summary table regarding clamping)

Post-Blend Color clamping is available only if Blending is enabled.

If Blending is enabled, the clamping of blending output color components is controlled by **Post-Blend Color Clamp Enable**. If ENABLED, the color components output from blending are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed at this point.

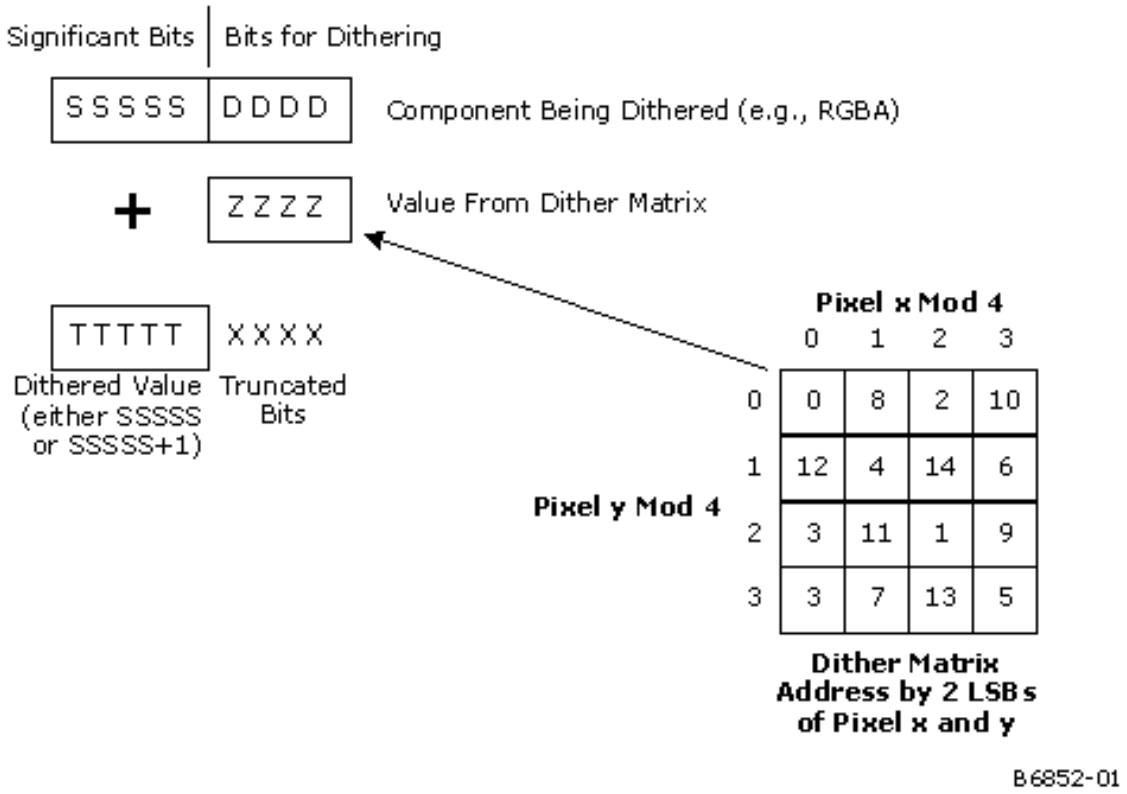
Regardless of the setting of **Post-Blend Color Clamp Enable**, when Blending is enabled color components will be automatically clamped to (at least) the rendertarget surface format range at this stage of the pipeline.

Dithering

Dithering is used to give the illusion of a higher resolution when using low-bpp channels in color buffers (e.g., with 16bpp color buffer). By carefully choosing an arrangement of lower resolution colors, colors otherwise not representable can be approximated, especially when seen at a distance where the viewer's eyes will average adjacent pixel colors. Color dithering tends to diffuse the sharp color bands seen on smooth-shaded objects.

A four-bit dither value is obtained from a 4x4 Dither Constant matrix depending on the pixel's X and Y screen coordinate. The pixel's X and Y screen coordinates are first offset by the **Dither Offset X** and **Dither Offset Y** state variables (these offsets are used to provide window-relative dithering). Then the two LSBs of the pixel's screen X coordinate are used to address a column in the dither matrix, and the two LSBs of the pixel's screen Y coordinate are used to address a row. This way, the matrix repeats every four pixels in both directions.

The value obtained is appropriately shifted to align with (what would be otherwise) truncated bits of the component being dithered. It is then added with the component and the result is truncated to the bit depth of the component given the color buffer format.



Dithering Process (5-Bit Example)

Logic Ops

The Logic Ops function is used to combine the incoming "source" pixel color/alpha values with the corresponding "destination" color/alpha contained in the ColorBuffer, using a logic function.

The Logic Op function is enabled by the **LogicOp Enable** state variable. If DISABLED, this function is ignored and the incoming pixel values are passed through unchanged.

Programming Notes

Programming Note
Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
Logic Ops are supported on all blendable render targets and render targets with *INT formats.

The following table lists the supported logic ops. The logic op is selected using the **Logic Op Function** field in COLOR_CALC_STATE.

Logic Ops

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_CLEAR	all 0's
LOGICOP_NOR	NOT (S OR D)

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_AND_INVERTED	(NOT S) AND D
LOGICOP_COPY_INVERTED	NOT S
LOGICOP_AND_REVERSE	S AND NOT D
LOGICOP_INVERT	NOT D
LOGICOP_XOR	S XOR D
LOGICOP_NAND	NOT (S AND D)
LOGICOP_AND	S AND D
LOGICOP_EQUIV	NOT (S XOR D)
LOGICOP_NOOP	D
LOGICOP_OR_INVERTED	(NOT S) OR D
LOGICOP_COPY	S
LOGICOP_OR_REVERSE	S OR NOT D
LOGICOP_OR	S OR D
LOGICOP_SET	all 1's

Buffer Update

The Buffer Update function is responsible for updating the pixel's Stencil, Depth and Color Buffer contents based upon the results of the Stencil and Depth Test functions. Note that Kill Pixel and/or Alpha Test functions may have already discarded the pixel by this point.

Stencil Buffer Updates

If and only if stencil testing is enabled, the Stencil Buffer is updated according to the **Stencil Fail Op**, **Stencil Pass Depth Fail Op**, and **Stencil Pass Depth Pass Op** state (or their backface counterparts if **Double Sided Stencil Enable** is ENABLED and the pixel is from a back-facing object) and the results of the Stencil Test and Depth Test functions.

Stencil Fail Op and **Backface Stencil Fail Op** specify how/if the stencil buffer is modified if the stencil test fails. **Stencil Pass Depth Fail Op** and **Backface Stencil Pass Depth Fail Op** specify how/if the stencil buffer is modified if the stencil test passes but the depth test fails. **Stencil Pass Depth Pass Op** and **Backface Stencil Pass Depth Pass Op** specify how/if the stencil buffer is modified if both the stencil and depth tests pass. The operations (on the stencil buffer) that are to be performed under one of these (mutually exclusive) conditions is summarized in the following table.

Stencil Buffer Operations

Stencil Operation	Description
STENCILOP_KEEP	Do not modify the stencil buffer
STENCILOP_ZERO	Store a 0

Stencil Operation	Description
STENCILOP_REPLACE	Store the *StencilTestReference* reference value
STENCILOP_INCRSAT	Saturating increment (clamp to max value)
STENCILOP_DECRSAT	Saturating decrement (clamp to 0)
STENCILOP_INCR	Increment (possible wrap around to 0)
STENCILOP_DECR	Decrement (possible wrap to max value)
STENCILOP_INVERT	Logically invert the stencil value

Any and all writes to the stencil portion of the depth buffer are enabled by the **Stencil Buffer Write Enable** state variable.

When writes are enabled, the **Stencil Buffer Write Mask** and **Backface Stencil Buffer Write Mask** state variables provide an 8-bit mask that selects which bits of the stencil write value are modified. Masked-off bits (i.e., mask bit == 0) are left unmodified in the Stencil Buffer.

Programming Note	
Context:	Stencil Buffer Updates
The Stencil Buffer can be written even if depth buffer writes are disabled via Depth Buffer Write Enable	

Depth Buffer Updates

Any and all writes to the Depth Buffer are enabled by the **Depth Buffer Write Enable** state variable. If there is no Depth Buffer, writes must be explicitly disabled with this state variable, or operation is UNDEFINED.

If depth testing is disabled or the depth test passed, the incoming pixel's depth value is written to the Depth Buffer. If depth testing is enabled and the depth test failed, the pixel is discarded – with no modification to the Depth or Color Buffers (though the Stencil Buffer may have been modified).

Color Gamma Correction

Computed RGB (not A) channels can be gamma-corrected prior to update of the Color Buffer.

This function is automatically invoked whenever the destination surface (render target) has an SRGB format (see surface formats in *Sampling Engine*). For these surfaces, the computed RGB values are converted from gamma=1.0 space to gamma=2.4 space by applying a ^(2.4) exponential function.

Color Buffer Updates

Finally, if the pixel has not been discarded by this point, the incoming pixel color is written into the Color Buffer. The **Surface Format** of the color buffer indicates which channel(s) are written (e.g., R8G8_UNORM are written with the Red and Green channels only). The **Color Buffer Component Write Disables** from the Color Buffer's SURFACE_STATE provide an independent write disable for each channel of the Color Buffer.

Pixel Pipeline State Summary

COLOR_CALC_STATE

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_DEPTH_STENCIL_STATE_POINTERS

3DSTATE_DEPTH_STENCIL_STATE_POINTERS does not exist for BDW+. It has been replaced by 3DSTATE_WM_DEPTH_STENCIL. (See Vol2a.11 3D Pipeline – Windower for details.)

COLOR_CALC_STATE

COLOR_CALC_STATE

DEPTH_STENCIL_STATE

DEPTH_STENCIL_STATE does not exist on BDW+. It has been replaced by 3DSTATE_WM_DEPTH_STENCIL. (See *3D Pipeline – Windower* for details).

BLEND_STATE

BLEND_STATE

Signal	CC_INT::AlphaTestEnable
Description	AlphaTestEnable
Formula	= BLEND_STATE::AlphaTestEnable && !3DSTATE_WM_HZ_OP::DepthBufferResolveEnable && !3DSTATE_WM_HZ_OP::DepthBufferClear && !3DSTATE_WM_HZ_OP::StencilBufferClear

Signal	CC_INT::AlphaToCoverageEnable
Description	AlphaToCoverageEnable
Formula	= BLEND_STATE::AlphaToCoverageEnable && !3DSTATE_PS_EXTRA::PixelShaderDisableAlphaToCoverage

CC_VIEWPORT

CC_VIEWPORT

Other Pixel Pipeline Functions

Statistics Gathering

If **Statistics Enable** is set in 3DSTATE_WM, the PS_DEPTH_COUNT register (see Memory Interface Registers in Volume 1a, *GPU Overview*) is incremented once for each pixel (or sample) that passes the depth, stencil and alpha tests. Note that each of these tests is treated as passing if disabled. This count is accurate regardless of whether **Early Depth Test Enable** is set. To obtain the value from this register at a deterministic place in the primitive stream without flushing the pipeline, however, the PIPE_CONTROL command must be used. See Volume 2a, *3D Pipeline*, for details on PIPE_CONTROL.

MCS Buffer for Render Target(s)

Lossless Color Compression on Render target can be enabled for the purposes described below:

1. MMIO bit Cache Mode 1 (0x7004) register bit 5
2. MMIO bit Cache Mode 1 (0x7004) register bit 15
3. RT surface state (Auxiliary Surface Mode[2:0])

Note: Lossless Color Compression can only be applied to Surfaces which are TileY, TileYs, or TileYf.

The following table summarizes modes of operation related to the Lossless Color Compression on Render target:

Cache Mode 1 MMIO Bit 5 (Please refer to Vol 1c)	Auxiliary Surface Mode Surface State	Cache Mode 1 MMIO Bit 15 (Please refer to Vol 1c)	Operation
1	X	X	Normal mode of operation i.e. no MSAA compression and no color clear
0	AUX_NONE (0h)	X	Normal mode of operation i.e. no MSAA compression and no color clear
0	AUX_CCS_D (1h)	X	Depending on the Number of multi-samples, either MCS or CCS is enabled. No MSAA or Color Compression is enabled.
0	AUX_CCS_E (5h)	1	Depending on the Number of multi-samples, either MCS or CCS is enabled. For Number of multi-samples > 1 MSAA Compression is enabled. For Number of multi-samples = 1 Color Compression is disabled.
0	AUX_CCS_E (5h)	0	Depending on the Number of multi-samples, either MCS or CCS is enabled. For Number of multi-samples > 1 MSAA Compression is enabled. For Number of multi-samples = 1 Color Compression is enabled.

MSAA	Width of Clear Rect	Height of Clear Rect
2x	Ceil(1/8*width)	Ceil(1/2*height)
4X	Ceil(1/8*width)	Ceil(1/2*height)
8X	Ceil(1/2*width)	Ceil(1/2*height)
16X	width	Ceil(1/2*height)

- **MSAA Compression:** Multi-sample render target is bound to the pipeline and MSAA compression feature is enabled. In this case, MCS buffer stores the information required for MSAA compression algorithm. The size and layout of the MCS buffer is based on per-pixel RT. For 4X and 8X MSAA, MCS buffer element is 8bpp and 32bpp respectively. Height, width, and layout of MCS buffer in this case must match with Render Target height, width, and layout. MCS buffer is tiledY. When MCS buffer is enabled and bound to MSRT, it is required that it is cleared prior to any rendering. A clear value can be specified optionally in the surface state of the corresponding RT. Clear pass for this case requires that scaled down primitive is sent down with upper left coordinate to coincide with actual rectangle being cleared. For MSAA, clear rectangle's height and width need to as show in the following table in terms of (width,height) of the RT.
- **Fast Color Clear:** When non multi-sample render target is bond to the pipeline and MSC buffer is enabled, MCS buffer is used as an intermediate (coarse granular) buffer per RT. Hence, MCS buffer is used to improve render target clear. When MCS is buffer is used for color clear of non-multisampler render target, the following restrictions apply:

Color Clear of Non-MultiSampler Render Target Restrictions

<table>
<tr><th>Restrictions</th></tr>
<tr><td>Support is limited to tiled render targets.</td></tr>
<tr><td>Mip-mapped and arrayed surfaces are supported with MCS buffer layout with these alignments in the RT space: Horizontal Alignment = 128 and Vertical Alignment = 64.</td></tr>
<tr><td>MCS and Lossless compression is supported for TiledY/TileYs/TileYf non-MSRTs only.</td></tr>
<tr><td>Clear is supported only on the full RT; i.e., no partial clear or overlapping clears.</td></tr>
<tr><td>The following table describes the RT alignment:
<table>
<tr><th>TiledY RT CL</th><th>Pixels</th><th>Lines</th></tr>
<tr><td>bpp</td><td></td><td></td></tr>
<tr><td>32</td><td>8</td><td>4</td></tr>
<tr><td>64</td><td>4</td><td>4</td></tr>
<tr><td>128</td><td>2</td><td>4</td></tr>
<tr><th>TiledX RT CL</th><th></th><th></th></tr>
<tr><td>bpp</td><td></td><td></td></tr>
<tr><td>32</td><td>16</td><td>2</td></tr>
<tr><td>64</td><td>8</td><td>2</td></tr>
<tr><td>128</td><td>4</td><td>2</td></tr>
</table>
</td></tr>
</table>

Restrictions
MCS buffer for non-MSRT is supported only for RT formats 32bpp, 64bpp, and 128bpp.
Clear pass must have a clear rectangle that must follow alignment rules in terms of pixels and lines as shown in the table below. Further, the clear-rectangle height and width must be multiple of the following dimensions. If the height and width of the render target being cleared do not meet these requirements, an MCS buffer can be created such that it follows the requirement and covers the RT.
Clear rectangle must be aligned to two times the number of pixels in the table shown below due to 16X16 hashing across the slice.

TiledY RT	Pixels	Lines
bpp		
32	128	64
64	64	64
128	32	64

To optimize the performance MCS buffer (when bound to 1X RT) clear similarly to MCS buffer clear for MSRT case, clear rect is required to be scaled by the following factors in the horizontal and vertical directions:

MCS CL for TiledY RCC	Horizontal Scale Down Factor	Vertical Scale Down Factor
bpp		
32	64	32
64	32	32
128	16	32

The following SW requirements for MCS buffer clear functionality apply in addition to the general SW requirements listed below:

- For non-MSRTs, loss less compression of render targets is supported for 32, 64, and 128 bpp surfaces as described in the Shared Functions Data Port under Render Target Write. Lossless RT compression can be enabled for each RT surface with auxiliary surface control bits as described in the Surface State. This feature changes the MSC layout for various tiled and BPP formats as described in above sections. Fast clear, render, and resolve operations are fundamentally the same. Since Sampler support reading this auxiliary (MCS) buffer for non-MSRTs, resolve passes can be avoided in the cases when fast cleared and possibly compressed RTs are consumed by the sampler.
- SW does not need to compile any PS for clear and resolve passes but must ensure that PS dispatch enable bit is set.
- Any transition from any value in {Clear, Render, Resolve} to a different value in {Clear, Render, Resolve} requires end of pipe synchronization.

The following are the general SW requirements for MCS buffer clear functionality:

- At the time of Render Target creation, SW needs to create clear-buffer, i.e., MCS buffer.

- At the clear time, clear value for that RT must be programmed and clear enable bit must be set in the surface state of the corresponding RT.
- SW must clear the RT with setting a RT clear bit set in the PS state during the clear pass as described in the following sub-section.
- Since only one RT is bound with a clear pass, only one RT can be cleared at a time. To clear multiple RTs, multiple clear passes are required.
- If Software wants to enable Color Compression without Fast clear, Software needs to initialize MCS with zeros.
- Lossless compression and CCS initialized to all F (using HW Fast Clear or SW direct Clear) on the same surface is not supported:
- Before binding the "cleared" RT to texture OR honoring a CPU lock OR submitting for flip, SW must ensure a resolve pass. Such a resolve pass is described in the following sub-section.

Render Target Fast Clear

Fast clear of the render target is performed by setting the **Render Target Fast Clear Enable** field in 3DSTATE_PS and rendering a rectangle The size of the rectangle is related to the size of the MCS.

The following is required when performing a render target fast clear:

- The render target(s) is/are bound as they normally would be, with the MCS surface defined in SURFACE_STATE.
- A rectangle primitive of the same size as the MCS surface is delivered.
- The pixel shader kernel requires no attributes, and delivers a value of 0xFFFFFFFF in all channels of the render target write message The replicated color message should be used.
- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable,** and **Alpha Test Enable** must all be disabled.
- After Render target fast clear, pipe-control with color cache write-flush must be issued before sending any DRAW commands on that render target.

Render Target Resolve

If the MCS is enabled on a non-multisampled render target, the render target must be resolved before being used for other purposes (display, texture, CPU lock) The clear value from SURFACE_STATE is written into pixels in the render target indicated as clear in the MCS. This is done by setting the **Render Target Resolve Enable** field in 3DSTATE_PS and rendering a full render target sized rectangle. Once this is complete, the render target will contain the same contents as it would have had the rendering been performed with MCS surface disabled. In a typical usage model, the render target(s) need to be resolved after rendering and before using it as a source for any consecutive operation.

When performing a render target resolve, PIPE_CONTROL with end of pipe sync must be delivered.

A rectangle primitive must be scaled down by the following factors with respect to render target being resolved:

MSC CL for TiledY RCC	Horizontal scale down factor	Vertical scale down factor
bpp		
32	64	32
64	32	32
128	16	32

Programming Note	
Context:	Render Target Resolve

- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable,** and **Alpha Test Enable** must all be disabled.
- This render target resolve procedure is not supported on multisampled render targets. Unresolved multisampled render targets are directly supported by the sampling engine, which resolves clear values in addition to decompressing the surface This applies to both *ld2dms* and *sample2dms* messages.

Media GPGPU Pipeline

This section of the BSpec discusses Programming the GPGPU Pipeline, Thread Group Tracking, Generic Media, and other related topics.

Media GPGPU Pipeline]

This section of the BSpec discusses Programming the GPGPU Pipeline, Thread Group Tracking, Generic Media, and other related topics.

Programming the GPGPU Pipeline

1. In MEDIA_VFE_STATE choose whether to set DW2.6 Bypass Gateway Control. Usually this will be set, allowing the gateway to be used without OpenGateway/CloseGateway.
2. Set the virtual pools of EUs via MEDIA_POOL_STATE.
3. Set up interface descriptor with # of threads in barrier. The barrier id is not specified here because can Gen7 automatically assigns barriers to thread groups when they are free. The amount of CURBE data to deliver per thread dispatch is set in the interface descriptor.
4. Set up CURBE with thread ids and common data for all thread dispatches in the thread group.
5. Set up a GPGPU_WALKER command or a set of GPGPU_OBJECT commands with the thread group ids to dispatch the threads. The CURBE data is sent in sections for each thread dispatch in the thread group; a new thread group starts sending the CURBE data from the beginning of the buffer.

Note: Gen7 can either have the barriers and SLM automatically managed by hardware or specified by software. Mixing software managed and hardware managed in the same set of threads is allowed, but may cause stalls if there is an allocation conflict.

Note: When using GPGPU_OBJECT, finish dispatching a thread group before starting a different one.

The kernel should handle the barriers as follows:

The BarrierMsg message contains the barrier id and a way to reprogram the barrier count. The barrier count reprogram is not normally used for GPGPU workloads. When all threads in the group have reached the barrier, the gateway returns a notification bit 0.

The kernel must wait for the barrier to finish with a WAIT N0.

If a GPGPU kernel is going to access the Render Cache then the Fixed Function DOP Clock Gate Disable should be set to prevent the clock gating of the DAP used to access the Render Cache. This should be reset as soon as all the kernels that are accessing the Render Cache are finished and the Render cache flushed.

GPGPU Thread Limits

GPGPU requires 1024 SIMD channels to be available for a maximum size thread group. In a HSW/GT2 system with 10 EUs per subslice, each with 7 hardware threads, this means that a maximum size thread group will fit in a subslice if SIMD16 instructions are used, but not if SIMD8 are used. These limits can be circumvented for thread groups which do not need access to a barrier or SLM, in which case the thread group can cross sub-slices.

The Configurations section should be referenced to determine what SIMD is required to fit in the subslice of the targeted configuration.

GPGPU Commands

This section contains various commands for GPGPU, including a number of them shared with media mode.

MEDIA_VFE_STATE with varying definitions for different generations and projects:

MEDIA_VFE_STATE

MEDIA_CURBE_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Interface Descriptor Data payload as pointed to by the Interface Descriptor Data Start Address, with varying definitions for different generations and projects:

INTERFACE_DESCRIPTOR_DATA

MEDIA_STATE_FLUSH

GPGPU_WALKER

GPGPU Command Workarounds

GPGPU Commands have some subtle programming restrictions and workarounds to be aware of, as described below.

GPGPU Indirect Thread Dispatch

Indirect thread dispatch allows one thread group to control the group size of a following thread group.

This is the sequence of commands in the ring buffer:

GPGPU_OBJECT/WALKER	// Either a set of objects or a walker to dispatch a thread group which will write the next group's properties to memory.
MI_FLUSH	// Make sure the thread group has finished executing.
MEDIA_CURBE_LOAD	// Load the thread ids for new group.
MI_LOAD_REGISTER_MEMORY	// Load the indirect MMIO GPGPU registers from the mem written by the previous group.
GPGPU_WALKER (indirect)	// A walker with the indirect bit set.

The first thread group writes this data to memory:

1. The thread ids delivered in the CURBE - written where the following MEDIA_CURBE_LOAD will read them.
2. The GPGPU_WALKER parameters are written to memory where the MI_LOAD_REGISTER_MEMORY will read them.
    a. GPGPU_DISPATCHDIMX - the X dimension of the number of thread groups to dispatch in:

DWord
7

    b. GPGPU_DISPATCHDIMY - the Y dimension of the number of thread groups to dispatch in:

DWord
10

    c. GPGPU_DISPATCHDIMZ - the Z dimension of the number of thread groups to dispatch in:

DWord
12

See vol1c Memory Interface and Command Stream for the MMIO register addresses and formats.

If any of the indirect registers are set to 0 then no work will be dispatched.

GPGPU Context Switch

The GPGPU pipeline supports interruption of GPGPU workloads on thread group boundaries. This is needed for general purpose GPGPUs that are so large that there is a risk of the display becoming non-responsive if the work cannot be interrupted for other jobs.

A workload is interrupted with the MI_ARB_CHECK command with the UHPTR register. The MI_ARB_CHECK command is placed throughout the command buffer. The driver updates the UHPTR register when a new context is needed; MI_ARB_CHECK checks for this and reprograms the head and tail pointers to the new batch of commands. The driver waits for the preemption to occur without going into RS2.

The GPGPU needs to modify this to allow a GPGPU_WALKER command to be interrupted. This is done by following each GPGPU_WALKER command with a MEDIA_STATE_FLUSH. This causes the CS to stop fetching commands until either the command completes or until the UHPTR valid bit is set.

GPGPU workloads can be dispatched with either GPGPU_OBJECT commands or GPGPU_WALKER commands. In the case of GPGPU_OJBECT, the MEDIA_STATE_FLUSH/ MI_ARB_CHECK pair must be placed in the batch buffer at thread group boundaries, since preemption cannot occur with a thread group partially dispatched. GPGPU_WALKER commands can dispatch multiple thread groups, in this case the MEDIA_STATE_FLUSH/ MI_ARB_CHECK follows each GPGPU_WALKER and the hardware takes care of noticing the UHPTR update and stopping at the next thread group boundary.

The commands in the batch buffer will look something like this:

Command Ring	Notes
MI_SET_CONTEXT	Go to GPGPU context
MI_BATCH_BUFFER_START	If new context, set address to top of batch. Otherwise, address needs to be set to the command preempted (given in the HWSP). The GP GPGPU bit must be set.

Command Batch	Notes
GPGPU_OBJECT	
GPGPU_OBJECT	
...	(more threads forming a complete thread group)
MEDIA_STATE_FLUSH	Check for preemption at thread group boundary. “Preemption” defined by the UHPTR valid bit set.
MI_ARB_CHECK	Move the head only if UHPTR valid bit is set.
...	
GPGPU_WALKER	
MEDIA_STATE_FLUSH	Check for preemption at thread group boundary internal to GPGPU_WALKER command. “Preemption” defined by the UHPTR valid bit set.
MI_ARB_CHECK	Move the head only if UHPTR valid bit is set.
...	

Command Batch	Notes
MI_BATCH_BUFFER_END	GPCS batch workload bit is cleared.

The context saved will consist of the state commands for VFE and a modified GPGPU_WALKER command with a new starting thread group id. On context restore, the commands are executed to start the GPGPU_WALKER where it left off before continuing with the rest of the command buffer.

An example software model for starting a preemption goes like this:

1. The UHPTR is reprogrammed to point to the current tail of the ring buffer.
2. Insert new commands:
    a. LRI to UHPTR to clear valid.
    b. Store Register to mem the preempted batch offset.
    c. Store Register to mem the preempted ring offset.
    d. Pipe_control notification.
    e. An MI_SET_CONTEXT to the new context is put into the ring.
3. Insert commands for new context. i.e. batch buffers.
4. Update Tail Pointer.

Programming Note	
Context:	GPGPU Context Switch
• 2-3 items above could happen during execution of a thread group so the HW may see the tail pointer updated before preemption starts. • The driver needs to turn off RC6 during items 1 and 4.	

GPGPU Context Switch

Context switch for BDW onwards allows the switch to take place in the middle of a thread group to provide better response time.

The command sequence for BDW has been simplified – MEDIA_STATE_FLUSH and MI_ARB_CHECK are now optional between commands for preemption to occur. MEDIA_STATE_FLUSH is now only needed before MEDIA_LOAD_CURBE commands to ensure CURBE is done being read before reloading it. The watermark bit in MEDIA_STATE_FLUSH is not needed, since the check is done automatically before a thread group is started.

Preemption can occur on commands listed here:

- MI_ARB_CHECK
- MEDIA_STATE_FLUSH
- PIPE_CONTROL
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT

- GPGPU_WALKER – Preemption can occur at any time, with thread groups partially complete; the system state is saved/restored for context save and restore
- MI_WAIT_FOR_EVENT
- MI_SEAMPHORE_WAIT

Messages to the Sampler must use headers (controlled by bit 19 of the Message Descriptor) when pre-emption is enabled.

See GPGPU Context Switch Workarounds for information about programming restrictions and workarounds.

Note that command preemption is not supported for MEDIA_OBJECT_* commands for MI_ARB_ON/OFF should be used to prevent preemption except at frame boundaries, where an MI_ARB_CHECK should be inserted.

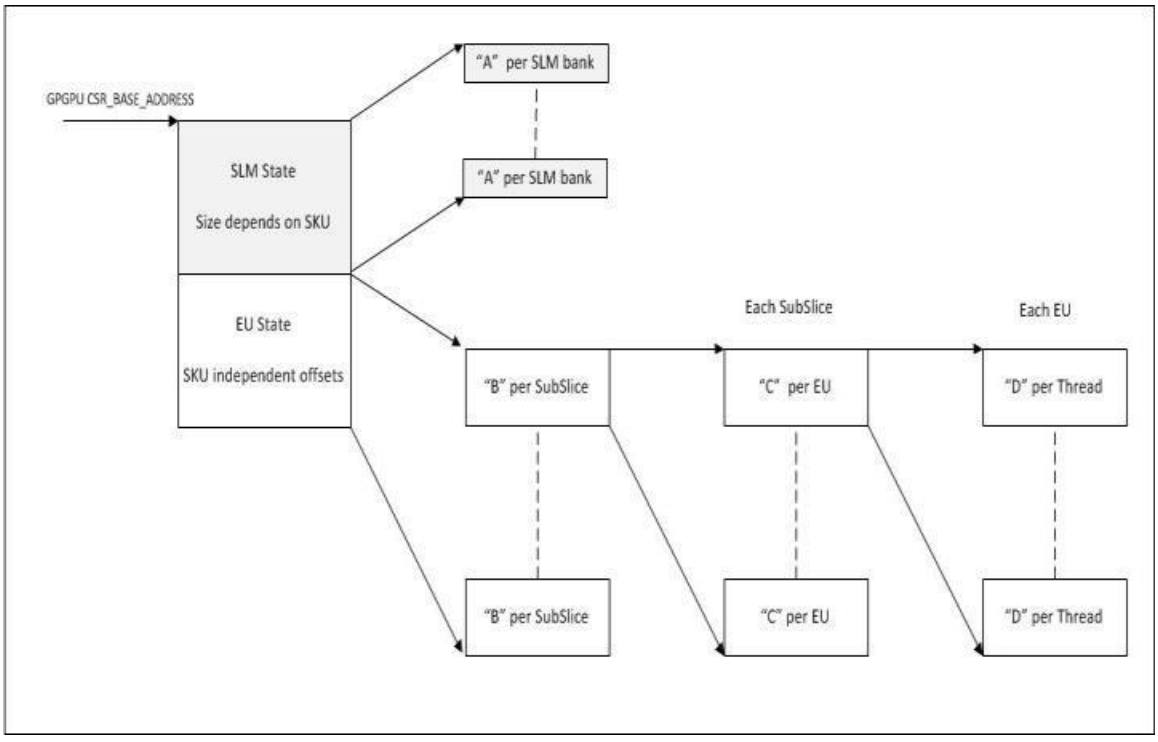
The memory map of the context image that is saved from GPGPU pipeline includes SLM and EU State. A contiguous space is allocated to save the SLM and EU State starting from the address provided in GPGPU_CSR_BASE_ADDRESS.

Maximum Upper Bound

In GPGPU context save/restore mode, hardware writes to this location and does NOT use the surface state or scratch space.

Memory Map of the GPGPU Context Image

The address offsets computed by hardware depend on the number of slices, subslices, EUs, and threads.



Base Address Calculation

Base Address = CSR_Base_Address + A* [Config.NumSlmBanks] + B*SubSliceId + C*EuId + D*ThreadId + Message_Offset

A = 0x10000 // 64K SLM per SubSlice

B = C * [Config.NumEusPerSubSlice]

C = D * [Config.NumThreadsPerEu]

D = 0x2000 // 8KB

Note: The slicenumber and EUIDs may require re-mapping such that a contiguous space is used with no gaps inbetween.

GPGPU Context Switch Workarounds

The GPGPU Context Switch support has some subtle programming restrictions and workarounds, as described below.

Media GPGPU Payload Limitations

There are 3 types of payload that the media/GPGPU instructions can have, but not all of them are allowed. The following table lists the legal combinations:

Workload	Commands	Data Stored
GPGPU	GPGPU_WALKER	CURBE
	GPGPU_WALKER	INDIRECT
Media(Legacy)	Media_Object	CURBE
	Media_Object	INDIRECT
	Media_Object	INLINE
	Media_Object	CURBE+INLINE
	Media_Object	CURBE+INDIRECT
	Media_Object	INLINE+INDIRECT
	Media_Object	CURBE+ INLINE+INDIRECT
	Media_Object_Walker	CURBE
	Media_Object_Walker	INLINE
	Media_Object_Walker	CURBE+INLINE
Media using Barrier/SLM	Media_Object_GRPID	CURBE
	Media_Object_GRPID	INDIRECT
	Media_Object_GRPID	INLINE
	Media_Object_Walker (with group id)	CURBE
	Media_Object_Walker (with group id)	INLINE

Indirect and CURBE payloads are fetched during thread dispatch from memory using the Dynamic State MOCS (specified in [Instruction] STATE_BASE_ADDRESS).

Synchronization of the Media/GPGPU Pipeline

The Media/GPGPU Pipeline is synchronized in the same way as the 3D pipeline using the PIPE_CONTROL command.

See the Bspec section on 3D pipe synchronization: vol2a 3D Pipeline - Overview > 3D Pipeline > Synchronization of the 3D Pipeline.

Mode of Operations

This section describes the dispatch and payload of the GPGPU threads.

GPGPU Mode

The general purpose (GPGPU) mode allows the Gen7 architecture to be used by general purpose parallel APIs:

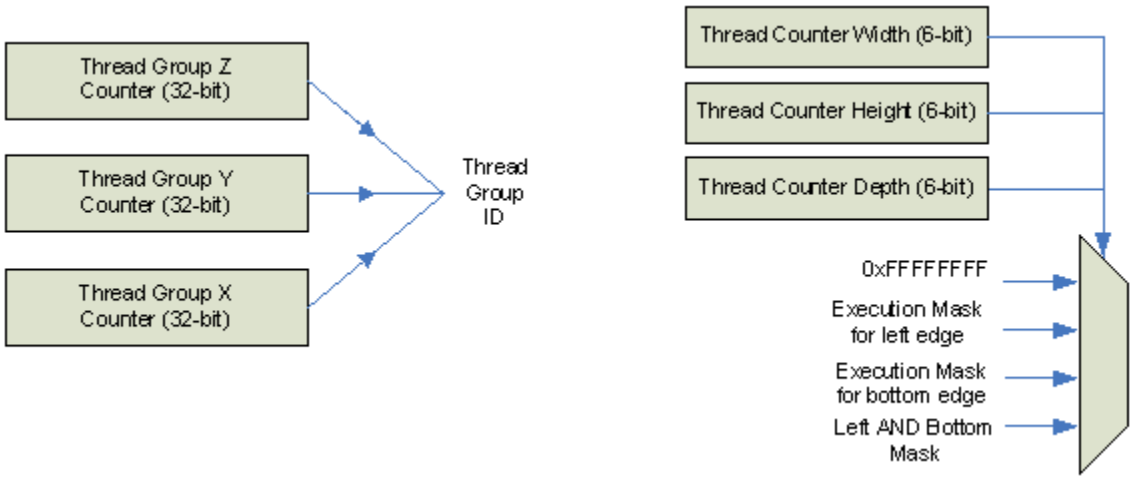
- OpenCL
- DX11 ComputeShader

This is similar to the Generic mode with additional support for automatic generation of threads, Shared Local Memory, and Barriers.

Automatic Thread Generation

A single GPGPU job may require thousands or even millions of GPU_OBJECT commands. Rather than create them separately, it would be better to generate them algorithmically. To do this a GPGPU_WALKER command is created.

Rather than modifying the Media Walker, a simple Thread Group Walker is created instead:



The X/Y/Z counters for the thread group will have an initial and maximum value. The thread group ID sent with each dispatch consists of these 3 numbers. These counters are 32 bits since the spec does not limit the size of the thread ID.

The 3 thread counters count the number of dispatches in a single thread group – up to 32 dispatches for SIMD32 or 64 dispatches for SIMD16/8. There are 3 thread counters in order to select the execution masks correctly – see the section *Execution Masks* . Each one is 6 bits to allow full flexibility of any dimension going to 64 while the rest do not increment.

A thread is generated each time one of the thread counters increment. When all the counters reach their maximum values, the thread group is done and the thread group counter can increment and start a new thread group. When the thread group X counter reaches its maximum it is reset to 0, and the Y counter is incremented.

The compiler determines how many SIMD channels are needed per thread group, and then decides how these are split among EU threads. The number of threads is programmed in the thread counter, and the SIMD mode (SIMD8/SIMD16/SIMD32) is specified in the GPGPU_WALKER command.

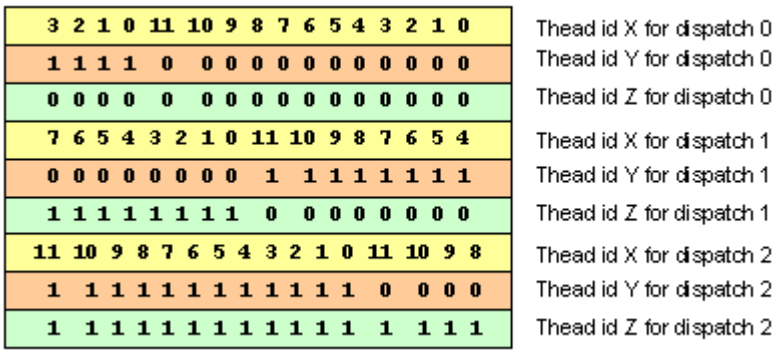
Thread Payload

The payload to each thread dispatched is:

1. A thread group id which identifies the group the set of threads belong to. This is in the form of a set of 3, 32-bit X/Y/Z values.
2. The set of X/Y/Z that form the thread ID for each channel. If Z is not used then only X/Y are needed.
3. The execution mask which indicates which channels are active.

Thread IDs form a 2D or 3D surface which has to be mapped into SIMD32, SIMD16 or SIMD8 dispatches. Rather than have the hardware force a particular mapping of thread IDs to channels, the mapping will be supplied by the compiler. The VFE will receive a simple count of the number of threads per thread group which will be used to count the number of dispatches. The thread IDs for all threads in a thread group are put in a constant buffer with the MEDIA_CURBE_LOAD command. A single set of thread IDs can be used repeatedly for all thread groups, since the thread IDs are the same for each thread group ID output by the GPGPU_WALKER.

The data required is up to the compiler, but here is an example set of payloads for a 2 Z x 2Y x 12 X and a SIMD16 dispatch. This thread group requires 3 dispatches:



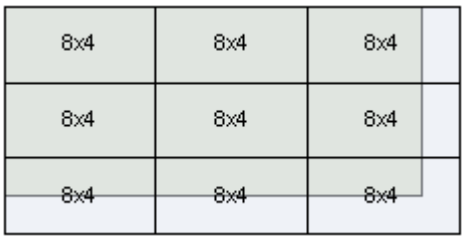
In this case the thread counter width would be programmed with a maximum value of 3 (since all the execution masks are all F, it doesn't matter how the thread counters are programmed as long as they count to 3 before finishing the thread group).

The first dispatch would tell the TS (who would tell the TD) that the payload starts at the beginning of the constant buffer and has a length of 3. The next dispatch would have a payload starting at constant_buffer_start + 3. The final dispatch payload starts at constant_buffer_start + 6. If there are more thread groups in the command they would get exactly the same payload – the only difference is the thread group ID (as well as a different barrier and shared local memory space).

Execution Masks

The number of channels required by the GPGPU job may not evenly fit into the number of SIMD channels. That can leave some channels idle. The execution mask is used to tell the hardware which channels are to be used.

A thread group is modeled as a 3D solid with each channel acting as one X/Y/Z point in the solid. This can take the form of a line with 1024 channels with X from 0 to 1023 and constant Y/Z, a square with X=0 to 32 and Y=0 to 32, or a cube with X=0 to 9, Y=0 to 9, Z=0 to 9. Software needs to determine how these shapes are mapped onto the 32 SIMD32 channels per dispatch (or 16 SIM16, etc). The mapping per thread is assumed to be a 2D square of channels such as 8x4, 16x2, 32x1. Below is a diagram of a 22x6 thread group that is mapped onto a set of 8x4 SIMD32 channels:



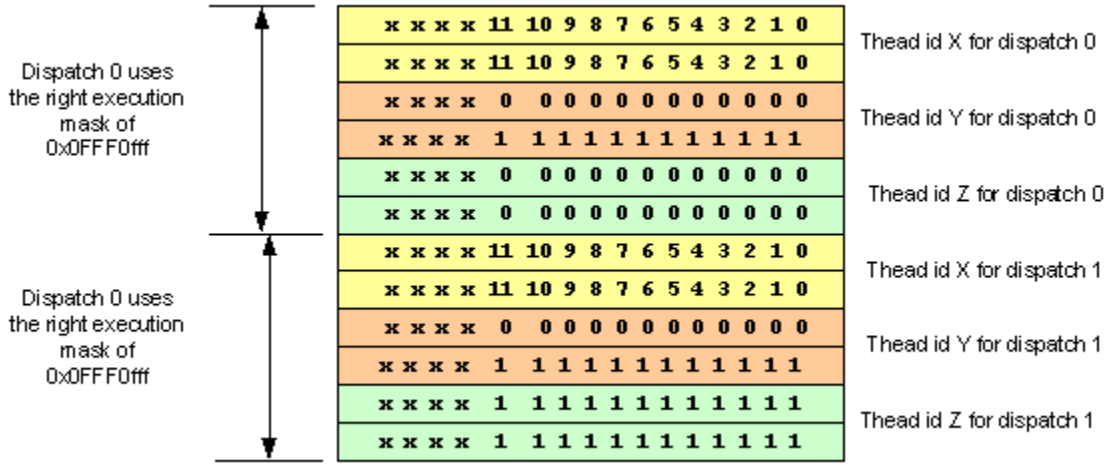
Note that the dispatches to the top and left have execution masks of all-F, while all the right edge dispatches have the same execution mask; likewise all the bottom edge dispatches have the same execution mask. The bottom right is the logical-AND of the right and bottom edge dispatches.

A 32-bit right and bottom mask is sent with the GPGPU_WALKER command, and the thread width, height and depth counters are used to determine when they are used (width, height and depth are used instead

of X/Y/Z, since it is not required that width = X – width and height are the two variables that are changing in a single SIMD dispatch even if they are Y and Z).

For each dispatch the width counter is incremented until it reaches the maximum – the dispatch with width=max will use the right execution mask. The height counter is then incremented and process repeated. If at any time the height counter = max then the execution mask is the bottom execution mask. When the height and width counters are both max then the dispatch will be the AND of the right and bottom and the depth counter will increment.

The same 2Z x 2Y x 12X thread group described above dispatched as SIMD32 with each dispatch delivering a 16X x 2Y shape would require 2 dispatches with empty bits in the right execution mask and all F in the bottom.



The width and height counter would have a maximum of 1, and the depth counter would have a maximum of 2. The two dispatches would use the AND of the two masks, but since the bottom mask is F it would be the same as just the right mask.

The execution masks can also be used when the software wants to pack the channels rather than lay them out in a regular pattern:

In this case the width counter can have a maximum of 2, and the height and depth counters with a maximum of 1. The first dispatch will use the bottom mask only (all-F) and the second will use the right AND bottom mask to remove the channels that are not used.

URB Management

The VFE manages the URB in GPGPU and generic/media modes.

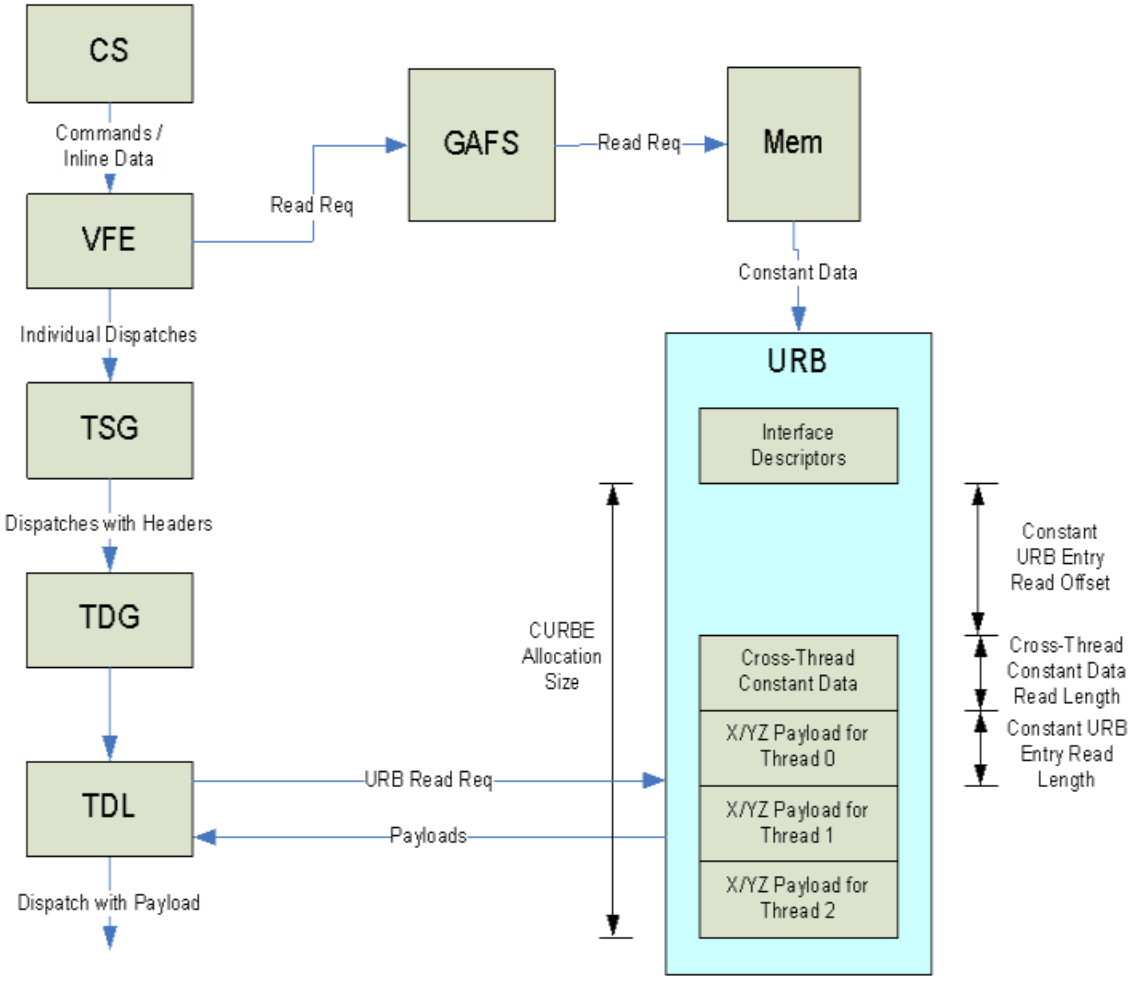
Description
The first 64 URB entries are reserved for the interface description, and CURBE data is placed after the IDs. URB handles are needed for indirect data and parent/child communication; when the VFE starts up it creates up to 128 handles by partitioning the remaining URB space into evenly spaced addresses and saving the resulting handles in a FIFO. The handles can then be treated just like ones created by the URBM - send to TD on dispatch and recovered on the handle return bus. MEDIA_VFE_STATE specifies the amount of CURBE space, the URB handle size and the number of URB handles. The driver must ensure that: ((URB_handle_size * URB_num_handle) – CURBE – 64) <= URB_allocation_in_L3.

Indirect Payload Storage

The GPGPU commands are extended to allow indirect input as an alternative to CURBE. The mechanism used for CURBE will be used in exactly the same way for indirect: the same offset is used which specifies what data is delivered to all threads, and then a count which specifies how much data is delivered per thread. A single indirect pointer points to both the Cross-Thread and Per-Thread Constant Data, which is stored in the URB. The position of the Cross-Thread and Per-Thread constants are swapped in the EU GRF.

To use indirect payload storage, the (URB Entry Allocation Size * Number of URB Entries) product must be enough to cover the sum of the Cross-Thread and Per-Thread Indirect data to be loaded. These state variables are set in the MEDIA_VFE_STATE command. The URB Entry Allocation must be equal or greater than th GPGPU_WALKER's Indirect Data Length. In general the maximum number of URB Entries that fills the URB space should be used to ensure that a shortage of handles doesn’t cause a performance problem.

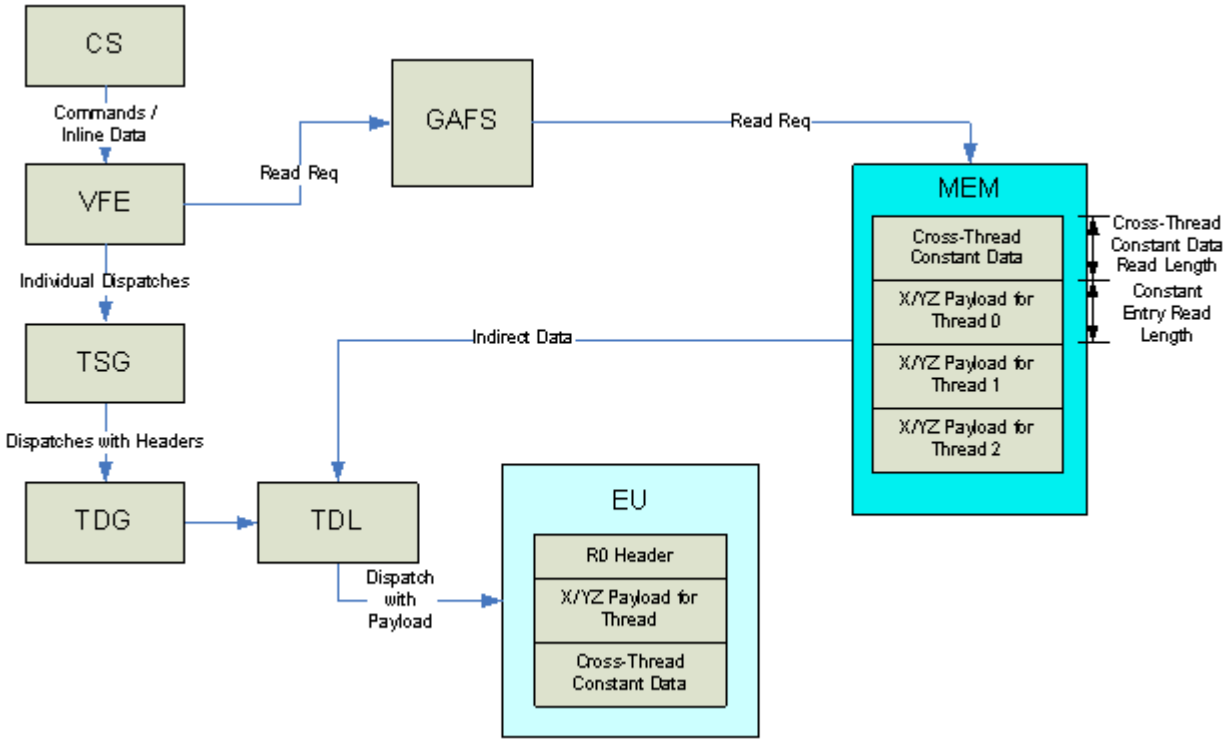
CURBE Payload



Example of CURBE command sequence:

```
MEDIA_STATE_FLUSH // Make sure dispatch is not accessing CURBE
MEDIA_CURBE_LOAD
GPGPU_WALKER
MEDIA_STATE_FLUSH
MEDIA_CURBE_LOAD
GPGPU_WALKER
```

Figure for Indirect Payload



Example of Indirect Command Sequence:

```
GPGPU_WALKER // Indirect pointer included in command
GPGPU_WALKER // No need to flush between commands for CURBE.
```

The differences between CURBE and Indirect are:

- The indirect uses a 32-bit memory pointer for the start address.
- The Constant URB Entry Read Offset is not used.
- The Cross-Thread Constant Data Read Length and the Constant Entry Read Length is multiplied by 32 to convert it into bytes.
- The X/Y/Z payload in the EU GRF comes before the Cross-Thread Constant Data.

MEDIA_OBJECT_GRPID

GPGPU_OBJECT already allows barriers and SLM to be allocated, but does not allow the scoreboard to be used. We need a single command which allows both.

A new command called MEDIA_OBJECT_GRPID is created which is the same as the basic MEDIA_OBJECT command with the addition of a 32-bit group id and an end-of-group bit. The group id will be used to allocate barriers and SLM based on the Interface Descriptor.

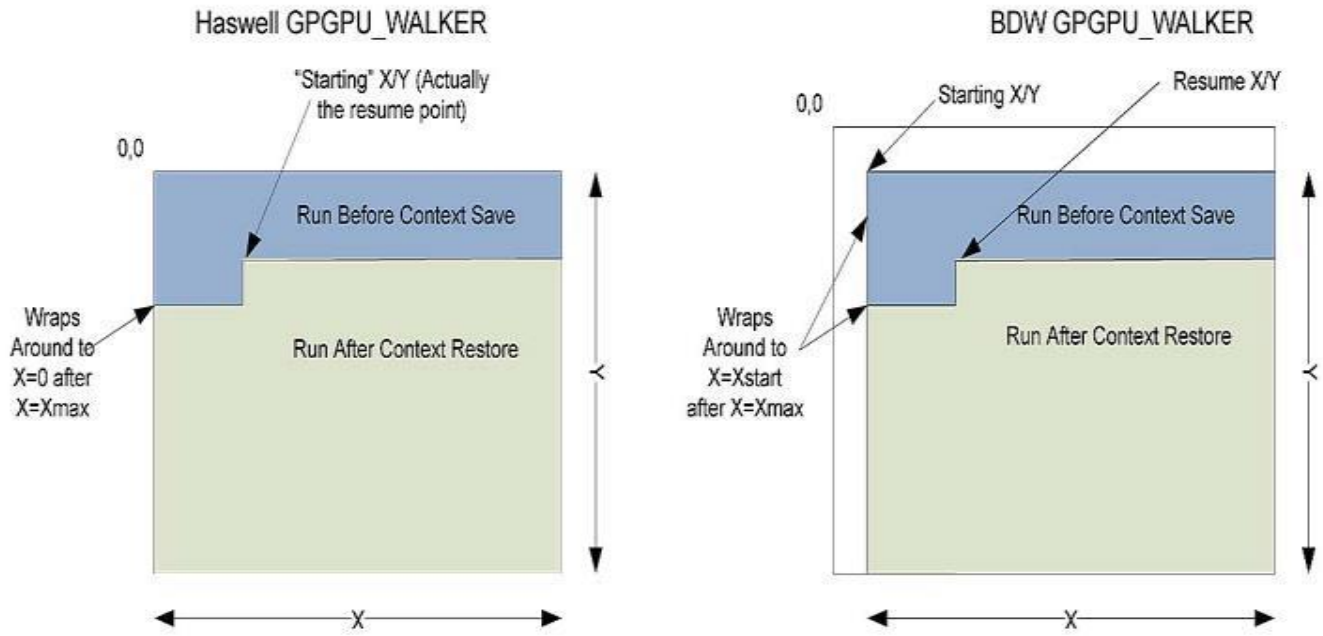
Restrictions that are currently in place for GPGPU would carry over to Media use of these features:

- All threads that use them must be on the same Tslice unless cross-slice barriers are enabled.

- We only dispatch a single group at a time – this is naturally supported by using only the most outer loop numbers for the in MEDIA_OBJECT_WALKER, but must be enforced by the programmer for MEDIA_OBJECT_GRPID.

Starting Offset for a Thread Group ID

The GPGPU_WALKER command has starting values for the Thread Group IDs that the walker generates per thread group, but they are designed to allow context save and restore rather than provide a true initial value. When walking a 2D set of groups in X/Y, when the X value reaches the maximum, the Y increments and the next X is 0, rather than going back to Xstart. This behavior is expected on context restore where the true initial values are 0,0 and the starting values are actually the resume values of the restore.



GPGPU Thread R0 Header

The RO header of the Thread Dispatch Payload for the GPGPU thread:

DWord	Bits	Description
R0.7	31:0	**Thread Group ID Z:** This field identifies the Z component of the thread group that this thread belongs to.
R0.6	31:0	**Thread Group ID Y:** This field identifies the Y component of the thread group that this thread belongs to.
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1K-byte aligned pointer to the scratch space (used for the GPGPU local memory space). Format = GeneralStateOffset[31:10]
	9	**GPGPU Dispatch.** Indicates that this dispatch is from the GPGPU pipe (see PIPELINE_SELECT command).
	8:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent threads (of any thread group). It is used to free up resources used by the thread upon thread completion. Format = U9.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4	Reserved: MBZ
	3:0	Indicates the stack memory size. Range = [0,11] indicating [1K bytes, 2M bytes] **Programming Note**: Exception handling on stack overflow is not supported when GPGPU mid-thread pre-emption is desired.
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 16-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two.

DWord	Bits	Description
R0.2	31	**Barrier ID MSB:** This field is bit[4] of the BarrierID; for full 5-bit barrier ids it should be combined with Barrier ID[3:0]. Format: U1
	30	Reserved: MBZ
	29	**Barrier Enable:** This field indicates that a barrier has been allocated for this kernel.
	28	**SLM Enable:** This field indicates that Shared Local Memory has been allocated for this kernel.
	27:24	**BarrierID:** This field indicates the barrier that this kernel is associated with. Format: U4
	14:11	Reserved: MBZ
	10	Reserved: MBZ
	3:0	Reserved: MBZ
R0.1	31:0	**Thread Group ID X:** This field identifies the X component of the thread group that this thread belongs to.
R0.0	31:30	Reserved: MBZ
	29:24	**Shared Local Memory Index:** Indicates the starting index for the shared local memory for the thread group. Format = U6
	23:16	Reserved: MBZ
	15:0	**URB Handle:** This is the URB handle indicating the URB space for use by the thread.

Cross-thread CURBE if present is in R1 and above, followed by the X/Y/Z thread id values for each channel in the thread.

Thread Group Tracking

The TSG needs to keep track of the threads outstanding in a group to know when the thread group barrier and Shared Local Memory can be reclaimed. This can be done by keeping a counter per active thread group (up to 16 per half-slice) which increments when a new thread is sent out and decremented when the thread retires. The assigned barrier ID (with half-slice bit) is unique per thread group and much smaller than the thread group ID and so will be used to keep track of the thread group instead.

Since TSL sends the thread retirement via the Message Channel rather than the thread retirement bus, the barrier ID used to identify the thread group can be sent at the same time. A CAM will then match the ID with the counter to decrement.

There is a potential corner case of a thread group without barriers being partly dispatched, then retiring before the rest of the thread group is sent. This should be OK, since the lack of barriers means that there are no dependencies between threads.

Shared Local Memory Allocation

The Shared Local Memory is a 64k block per half-slice in the L3 that must be shared between all thread groups on that half-slice. A new memory manager simular to the Scratch Space memory manager is used to allocate this space.

We are only dispatching threads from a single Interface Descriptor at a time. If a new Interface Descriptor is requested the pipe is drained and all shared memory recovered before starting to allocate new shared memory. This means that only a single size of shared memory needs to be supported at once.

For simplicity, only power-of-2 sizes from 4k to 64k are allowed. The thread request will specify how much is needed. The first thread of a Thread Group is marked as requiring a new shared local memory – if not the old Shared Local Memory offset is sent with the dispatch.

A simple set of 16-bits is used to allocate 4k shared memory, with fewer bits used for larger sizes. A priority encoder finds the first unused bit and the offset remembered as being associated with a particular barrier id. The barrier id is then used to track the thread group.

When the Thread Group Tracking indicates that a thread group is completely retired, that section of shared local memory can be reclaimed.

Software Managed Shared Local Memory

Software can optionally manage shared local memory. In this case, each thread command or thread group command will have the shared memory offset included – each command in a thread group must have the same offset, of couse. If the offset requested is still being used then the command is stalled until the thread group using that offset is done.

Hardware will track the usage of this section of shared memory as before, recording the offset as being used and recording it as being available after the thread group is done.

Automatic Barrier Management

Description
Since we have an automatic shared memory allocation it makes sense to make barrier management automatic too. Instead of the barrier id in the Interface Descriptor, there is now a thread count per thread group.
If a new thread group id comes in without a barrier allocated (checked with a CAM match across 32 barriers per subslice), the TSG picks a unused barrier and sends this count in a message to GWunit. It then needs to wait for an accept message back from GW before sending the dispatch to ensure that a barrier message doesn't arrive at the GW before the barrier is programmed. The barrier ID picked is sent with every dispatch from this thread group.
When the thread group tracker determines that a thread group has finished, the barrier becomes available to new thread groups.

Dispatch Payload

The payload for a general purpose thread will have to include the execution mask with a bit per 32-channel. SIMD16 and SIMD8 use the LSB bits of the execution mask. The 5-bit number transferred from VFE will be expanded to produce the 32-bit mask. This will use the Dmask currently used by the pixel shader dispatch in the transparent header.

Cross-Slice Barriers

Barriers in earlier projects were limited to communicating with EUs only in the local subslice. This works for GPGPU workloads, but Media workloads don't always fit in a subslice, and would be much improved with the ability to use barriers to synchronize across threads.

When a barrier is enabled with the global barrier bit set and SLM is disabled, the MEDIA_OBJECT_WALKER and MEDIA_OBJECT_GRPID can allow groups to cross slices. If SLM is enabled, then the restriction described in the following table applies:

Restriction
If SLM is enabled, then the group must stay on a single subslice or pool.

Each barrier will be marked as either local or global. If global it will force all subslices to use the same barrier id to lower gatecount in TSG – though this might cause a delay if a bunch of local barriers are allocated and no single global id is available across the needed subslices.

If the barrier is marked as global, the TSG will know that that group does not need to be forced into a single subslice. It will know from the interface descriptor how many threads are part of the barrier, but it will have to split this number among the subslices. TSG will dispatch as many threads as possible to a single subslice before going to the next to keep the number of subslices involved as low as possible. TSG will need to wait until there is enough threads available in the global barrier for the entire group before starting the dispatch so that it will know beforehand what to program the local subslice barrier to. Note that reprogramming the number of threads in a barrier in GW with the EU through the barrier message will not work in this case, since the EU can only reprogram the local GW.

Each subslice is programmed with the local number of threads and will work independently - when the local set of threads has been met, the GW will send a message to TSG. TSG has a simplified version of the

barrier logic - it has a count of how many GWs it expects to report per barrier (up to 8) and a count of how many have reported so far. It can use a simple bit-mask to record the GWs that have reported back so far. When all the gateways have reported, it will send a message back to each saying telling them to complete the barrier.

This scheme is slower than a normal barrier, but should be much faster than using atomics and polling to synchronize threads.

Programming Note	
Context:	Cross-Slice Barriers
When using cross-slice barriers, enough subslices must be available to run the group using the barrier - slice shutdown for power must be limited to ensure this.	

Generic Media

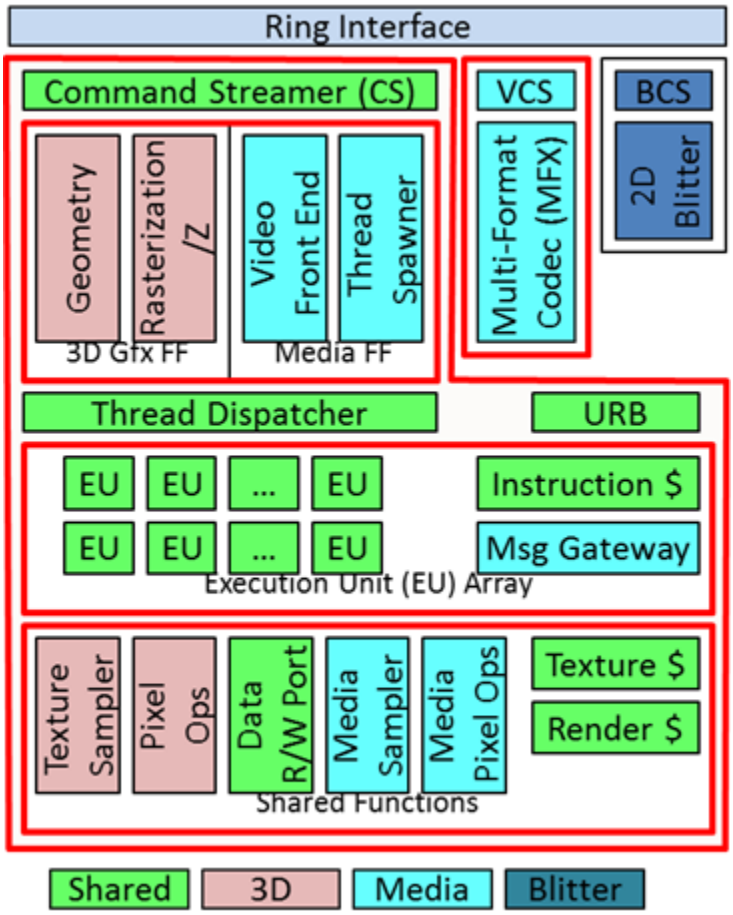
This introduction provides a brief overview of the Media product features. It includes Media functions, feature benefits, and how the features fit into graphics products as part of a whole solution.

Media product features include:

- Multi-format codec engine
- Video front end
- Media fixed functions
- Video encoding
- Video decoding
- Sampling

These product features support specific applications, such as interactive gaming, videogames, social media, virtual reality, and augmented reality.

The following block diagram shows the Main Render Engine, unified for 3D graphics and Media.



- **Fixed Function (FF) pipelines:** Provide thread generation and control.
- **3D graphics or Media FF:** Controls EU array at a given time. The EU (Execution Unit) array is shared between 3D and Media; ISA is optimized for both.
- **Shared functions:** Are accelerators to run filtered load, scatter, gather, and filter/blended store operations.
- **MFX:** Is a parallel codec engine that runs in a separate context.

Media and General Purpose Pipeline

Introduction

This section covers the programming details for the media (general purpose) fixed function pipeline. The *media pipeline* is positioned in parallel with the 3D fixed function pipeline. It provides media functions and has media specific fixed function capability. However, the fixed functions are designed to have the general capability of controlling the shared functions and resources, feeding generic threads to the Execution Units to be executed, and interacting with these generic threads during run time. The media pipeline can be used for non-media applications, and therefore, can also be referred to as the *general purpose pipeline*. For the rest of this chapter, we refer to this fixed function pipeline as the media pipeline, keeping in mind its general purpose capability.

Concurrency of the media pipeline and the 3D pipeline is not supported. In other words, only one pipeline can be activated at a given time. Switching between the two pipelines within a single context is supported using the MI_PIPELINE_SELECT command.

Following are some media application examples that can be mapped onto the media pipeline. All these applications are functional; however, the level of performance that can be achieved depends on the hardware configuration and is beyond the scope of this document.

Application
MPEG-2 decode acceleration with HWMC (e.g. DXVA HWMC interface)
MPEG-2 decode acceleration with IS/IDCT and forward (e.g. DXVA IDCT interface)
MPEG-2 decode acceleration with VLD and forward (e.g. DXVA VLD interface)
AVC decode acceleration with HWMC and forward including Loop Filter
VC1 decode acceleration with HWMC and forward including Loop Filter
Advanced deinterlace filter (motion detected or motion compensated deinterlace filter)
Video encode acceleration (with various level of hardware assistance)

Terminology

Term	Description
AVC	Advanced Video Coding. An international video coding standard jointly developed by MPEG and ITU. It is also known as H.264 (ITU), or MPEG-4 Part 10 (MPEG).
Child Thread	A thread corresponding to a leaf-node or a branch-node in a thread generation hierarchy. All thread originated from kernels running on the execution units are child threads.
EOB	End of Block. It is a 1-bit flag in the non-zero DCT coefficient data structure indicating the end of an 8x8 block in a DCT coefficient data buffer.
IDCT	Inverse Discrete Cosine Transform. It is the stage in the video decoding pipe between IQ and MC.
ILDB	In-loop Deblocking Filter – the deblocking filter operation in the decoding loop. It is a stage after MC in the video decoding pipe. It is required to support VC-1 Profile B video decoder acceleration.

Term	Description
IQ	Inverse Quantization. It is a stage in the video decoding pipe between IS and IDCT.
IS	Inverse Scan. It is a stage in the video decoding pipe between VLD and IQ. In this stage, a sequence of none-zero DCT coefficients are converted into a block (e.g. an 8x8 block) of coefficients. VFE unit has fixed functions to support IS for MPEG-2 and VC-1.
IT	Inverse Integer Transform. It is the stage in AVC or VC-1 video decoding pipe between IQ and MC.
MPEG	Motion Picture Expert Group. MPEG is the international standard body JTC1/SC29/WG11 under ISO/IEC that has defined audio and video compression standards such as MPEG-1, MPEG-2, and MPEG-4, etc.
MC	Motion Compensation. It is part of the video decoding pipe.
MVFS	Motion Vector Field Selection – a four-bit field selecting reference fields for the motion vectors of the current macroblock.
PRT	A persistent root thread in general stays in the system for a long period of time. It is normally a parent thread. Only one PRT is allowed in the system. Hardware is responsible for re-dispatching the incomplete PRT at context restore, and a PRT can continue operations from that previously left-over state.
Parent Thread	A thread corresponding to a root-node or a branch-node in thread generation hierarchy. A parent thread may be a root thread or a child thread depending on its position in the thread generation hierarchy.
Root Thread	A thread corresponding to a root-node in a thread generation hierarchy. In the general-purpose pipeline, all threads originated from VFE unit are root threads.
Synchronized Root Thread	A root thread that is dispatched by TS upon a 'dispatch root thread' message.
TS	Thread Spawner. It is the second (and the last) fixed function in the general-purpose pipeline.
Unsynchronized Root Thread	A root thread that is automatically dispatched by TS.
VC-1	VC-1 is the common name for the open video compression standard SMPTE 421M, adopted on April 3, 2006.
VFE	Video Front End. It is the first fixed function in the general-purpose pipeline.
VLD	Variable Length Decode. It is the first stage of the video decoding pipe that consists mainly of bit-wide operations. Hardware MPEG-2 VLD acceleration is supported in the VFE fixed function stage.

Hardware Feature Map in Products

The following table lists the hardware features in the media pipe.

Video Front End Features in Device Hardware

Features/ Device	
Generic Mode	**Y**
Root Threads	**Y**
Parent/Child Threads	**Y**
SRT (Synchronized Root Threads)	**Y**
PRT (Persistent Root Thread)	**Y**
Interface Descriptor Remapping	N
IS Mode (HW Inverse Scan)	N
VLD Mode (HW MPEG2 VLD)	N
AVC MC Mode	N
AVC IT Mode (HW AVC IT)	N
AVC ILDB Filter (in Data Port)	N
VC1 MC Mode	N
VC1 IT Mode (HW VC1 IT)	N
Stalling HW Scoreboard	**Y**
Non-stalling HW Scoreboard	**Y**
HW Walker	**Y**
HW Timer	**Y**
Pipelined State Flush	**Y**
HW Barrier	**Y**

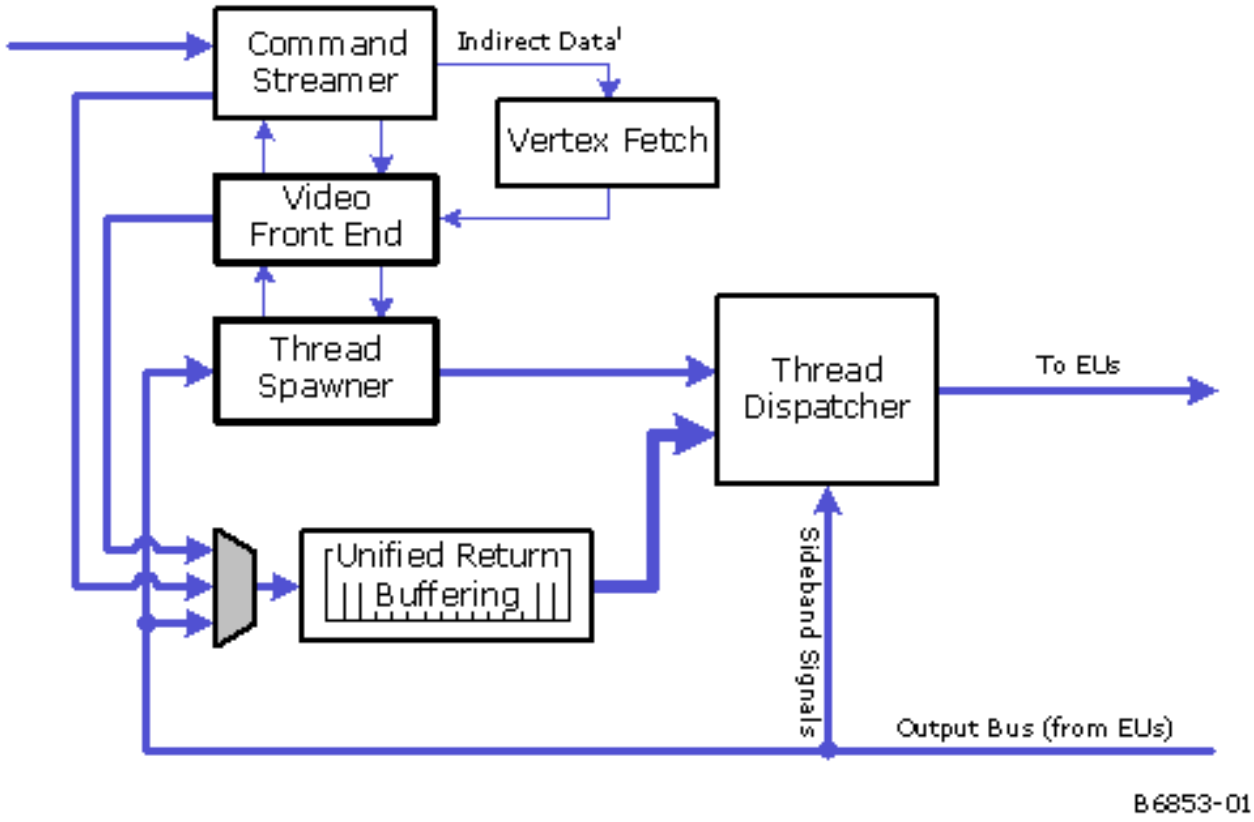
Media Pipeline Overview

The media (general purpose) pipeline consists of two fixed function units: Video Front End (VFE) unit and Thread Spawner (TS) unit. VFE unit interfaces with the Command Streamer (CS), writes thread payload data into the Unified Return Buffer (URB), and prepares threads to be dispatched through TS unit. VFE unit also contains a hardware Variable Length Decode (VLD) engine for MPEG-2 video decode. TS unit is the only unit of the media pipeline that interfaces to the Thread Dispatcher (TD) unit for new thread generation. It is responsible for spawning root threads (short for the root-node parent threads) originated from VFE unit and for spawning child threads (can be either a leaf-node child thread or a branch-node parent thread) originated from the Execution Units (EU) by a parent thread (can be a root-node or a branch-node parent thread).

The fixed functions, VFE and TS, in the media pipeline, in most cases, share the same basic building blocks as the fixed functions in the 3D pipeline. However, there are some unique features in media fixed functions as highlighted by the followings.

- VFE manages URB and only has write access to URB; TS does not interface to URB.
- When URB Constant Buffer is enabled, VFE forwards TS the URB Handler for the URB Constant Buffer received from CS.
- TS interfaces to TD; VFE does not.
- TS can have a message directed to it like other shared functions (and thus TS has a shared function ID), and it does not snoop the Output Bus as some other fixed functions in the 3D pipeline do.
- A root thread generated by the media pipeline can only have up to one URB return handle.
- If a root thread has a URB return handle, VFE creates the URB handle for the payload to initiating the root thread and also passes it alone to the root thread as the return handle. The root thread then uses the same URB handle for child thread generation.
- If URB Constant Buffer is enabled and an interface descriptor indicates that it is also used for the kernel, TS requests TD to load constant data directly to the thread's register space. For root thread, constant data are loaded after R0 and before the data from the other URB handle. For child thread, as the R0 header is provided by the parent thread, Thread Spawner splits the URB handles from the parent thread into two and inserts the constant data after the R0 header.
- A root thread must terminate with a message to TS. A child thread should also terminate with a message to TS.
- High streaming performance of indirect media object load is achieved by utilizing the large vertex cache available in the Vertex Fetch unit (of the 3D pipeline).

Top level block diagram of the Media Pipeline



Generic Mode

In the Generic mode, VFE serves as a conduit for general-purpose kernels fully configured by the host software. As there is no special fixed function logic used, the Generic mode can also be viewed as a 'pass-through' mode. In this mode, VFE generates a new thread for each MEDIA_OBJECT command. The payload contained in the MEDIA_OBJECT command (inline and/or indirect) is streamed into URB. The interface descriptor pointer is computed by VFE based on the interface descriptor offset value and the interface descriptor base pointer stored in the VFE state. VFE then forwards the interface descriptor pointer and the URB handle to TS to generate a new root thread. Many media processing applications can be supported using the Generic mode: MPEG-2 HWMC, frame rate conversion, advanced deinterface filter, to name a few.

Programming Media Pipeline

The Programming Media Pipeline is programmed with command sequences. The media hardware threads are created through the parameterized media walker. The dispatch of thread is controlled by a scoreboard mechanism.

Command Sequence

Media pipeline uses a simple programming model. Unlike the 3D pipeline, it does not support pipelined state changes. Any state change requires an MI_FLUSH or PIPE_CONTROL command. When programming the media pipeline, it should be cautious to not use the pipelining capability of the commands described in the Graphics Processing Engine chapter.

To emphasize the non-pipeline nature of the media pipeline programming model, the programmer should note that if any one command is issued in the "Primitive Command" step, none of the state commands described in the previous steps cannot be issued without preceding with a MI_FLUSH or PIPE_CONTROL command.

Note for : With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. The MEDIA_STATE_FLUSH serves as a fence for state change by flushing the VFE/TS front ends but not waiting for threads to retire.

The basic steps in programming the media pipeline are listed below. Some of the steps are optional; however, the order must be followed strictly. Some usage restrictions are highlighted for illustration purpose. For details, refer to the respective chapters for these commands.

Command Sequence

For , the media pipeline is further simplified with fixed functions like MPEG2 VLD and AVC/VC1 IT removed. The addition includes:

1. The CURBE command is now unique to the media pipeline.
2. The interface descriptors are delivered directly as a media state command instead of being loaded through indirect state.

The programming model requires the following steps:

Step 1: MI_FLUSH/PIPE_CONTROL:

- This step is mandatory.
- Multiple such commands in step 1 are allowed, but not recommended for performance reasons.

Step 2: State command PIPELINE_ SELECT:

- This step is optional. This command can be omitted if it is known that within the same context the media pipeline was selected before Step 1.
- Multiple such commands in step 2 are allowed, but not recommended for performance reasons.

Step 3: State commands configuring pipeline states:

- STATE_BASE_ADDRESS:
  - This command is mandatory for this step (i.e. at least one).
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
  - This command must precede any other state commands below.
  - Particularly, the fields **Indirect Object Base Address** and **Indirect Object Access Upper Bound** are used to control indirect Media object load in VF.
  - The fields **Dynamics Base Address** and **Dynamics Base Access Upper Bound** are used to control indirect Curbe and Interface Descriptor object load in VF.
  - **Note:** This command may be inserted before (and after) any commands listed in the previous steps (Step 1 and 2). For example, this command may be placed in the ring buffer while the others are put in a batch buffer.
- STATE_SIP:
  - This command is optional for this step. It is only required when SIP is used by the kernels.
- MEDIA_VFE_STATE:
  - This command is mandatory for this step (i.e. at least one).
  - This command cause destruction of all outstanding URB handles in the system. A new set of URB handles will be generated based on state parameters, no. of URB and URB length, programmed in VFE FF state.
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_CURBE_LOAD:
  - This command is optional.
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_POOL_STATE:
  - This command is advisable as it sets the virtual pool sizes of EUs to contain workgroup execution.
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_INTERFACE_DESCRIPTOR_LOAD:
  - This command is mandatory for this step (i.e. at least one).
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.

Step 4: Primitive commands:

- MEDIA_OBJECT:
  - This step is optional, but it does not make practical sense to not issue media primitive commands after going through the previous steps to set up the media pipeline.
  - Multiple such commands in step 4 can be issued to continue processing media primitives.

With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. To support context switch for barrier groups, watermark and barrier dependencies are added to the MEDIA_STATE_FLUSH command. The usage of barrier group may have strict restriction that all threads belonging to a barrier group must all be present to avoid deadlock during context switch. Here are the example programming sequences to allow context switch.

Notes
The use of MEDIA_OBJECT_PRT and MI_ARB_ON_OFF are optional.

- MEDIA_VFE_STATE
- MEDIA_INTERFACE_DESCRIPTOR_LOAD
- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 1)
- MEDIA_OBJECT_PRT (with VFE_STATE_FLUSH set and PRT NEEDED set)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 1
- Several MEDIA_OBJECT command (for barrier group 1)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)
- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 2)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 2
- Several MEDIA_OBJECT command (for barrier group 2)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- ...
- MI_FLUSH

Commands for the GPGPU pipe (GPGPU_OBJECT and GPGPU_WALKER) should be separated from commands for the Media pipe (MEDIA_OBJECT*) by an MI_FLUSH.

Parameterized Media Walker

The Parameterized Media Walker is a hardware thread generation mechanism that creates threads associated with units in a generalized 2-dimensional space, for example, blocks in a 2D image. With a small number of unit step vectors, the walker can implement a large number of walking patterns as described hereafter. This command may provide functions that are normally handled by the host software, thus, may be used to simplify the host software and GPU interface.

The walker described herein is doubly nested, where essentially a “local” walker can perform a variety of 2-dimensional walking patterns and a “global” walker can perform similar 2-dimensional walking

patterns upon many local walkers. The local walker has 3 levels (outer, middle, and inner) while the global walker has 2 levels (outer and inner). Thus, the algorithm has 5-nested loops that modify local state based on user-defined unit step vectors.

The Walker's programmability is derived from:

- The walker traverses a unit-normalized surface. Some example unit sizes:
    - 1x1: Walking pixels
    - 4x4: Walking sub-blocks
    - 16x16: Walking macro-blocks
    - 32x16: Walking macro-block-pairs
- The use of unit step vectors to describe the motion at each of level of nesting
- Starting locations for the local and global walkers
- Block sizes of the local and global walker
- And a small number of special mode controls for the inner-most loop which are aimed at efficiently dividing an image into two balanced workloads for dual-slice designs.

Walker Parameter Description

The global and local loops are both described by the same four parameters:

- Resolution,
- Starting location,
- Outer unit vector,
- Inner unit vector

The local inner loop has some special modes that will be described later. A table of the user inputs and some example values are given below:

GLOBAL LOOP PARAMETERS							
Global Resolution		Global Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
120	68	0	0	32	0	0	32

LOCAL LOOP PARAMETERS							
Block Resolution		Local Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
32	32	0	0	1	0	-2	2

LOCAL INNER LOOP SPECIAL MODE SELECTS							
Dual Mode	Repel	Attract			ExtraSteps	X	Y
TRUE	FALSE	FALSE			1	0	1

It should be emphasized that the value of what a "unit" represents is implicitly defined by the user. In other words, the walker traverses a "unit normalized space" that is not inherently bound to pixel walking.

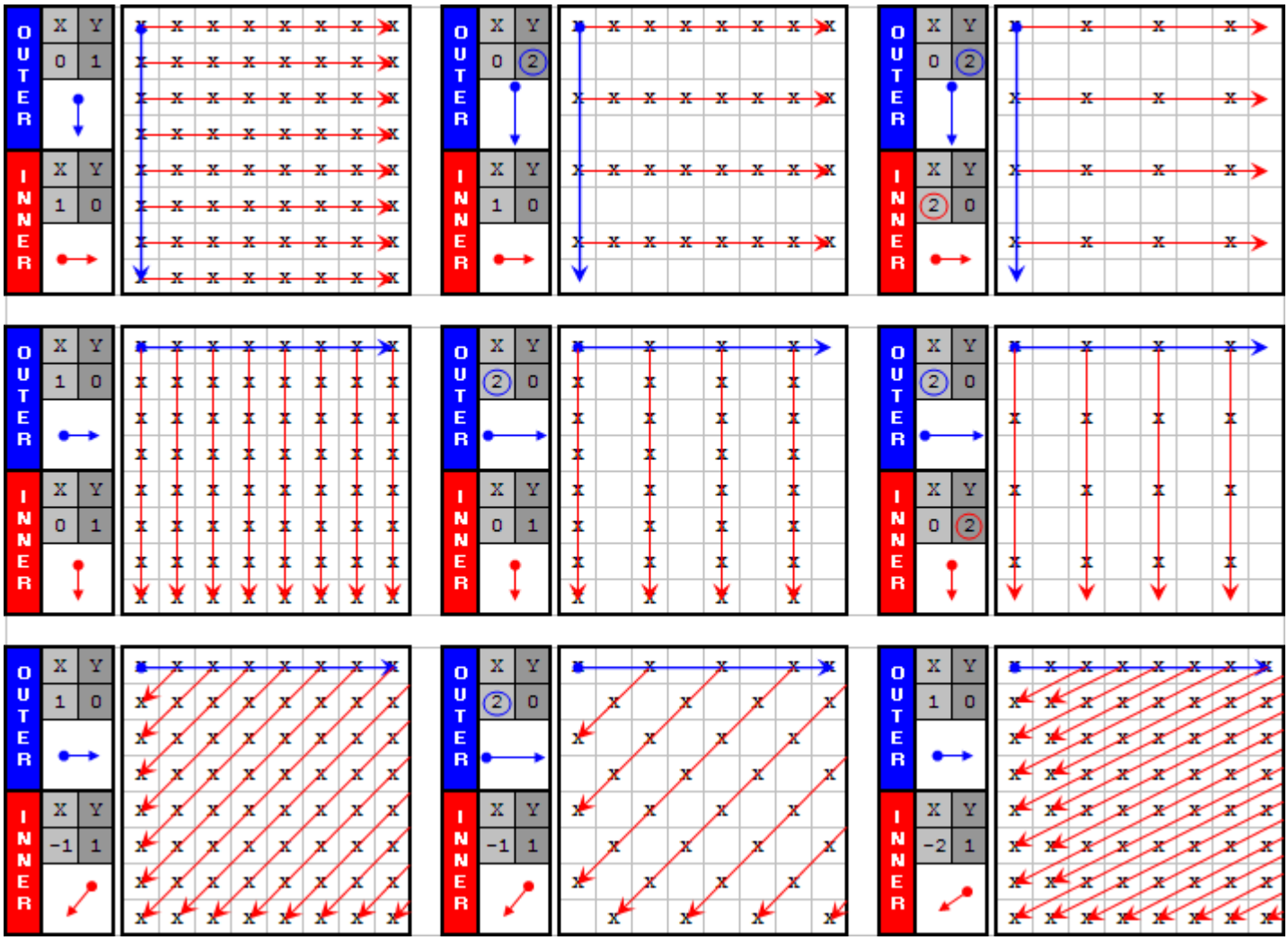
If the smallest unit of work the user wants to walk is a 4x3 block of pixels, you can program the inner loop to step (4,3) or (1,1):

- In the first case (4,3) the user is walking in units of pixels
- In the second case (1,1) the user is walking in units of 4x3 blocks of pixels.

It should be noted that hardware doesn't contain enough bits for pixel walking for pixel resolution like 1920x1088. The intended usage of the walker is for block walking whereas the block size is not relevant to the walker parameters.

Basic Parameters for the Local Loop

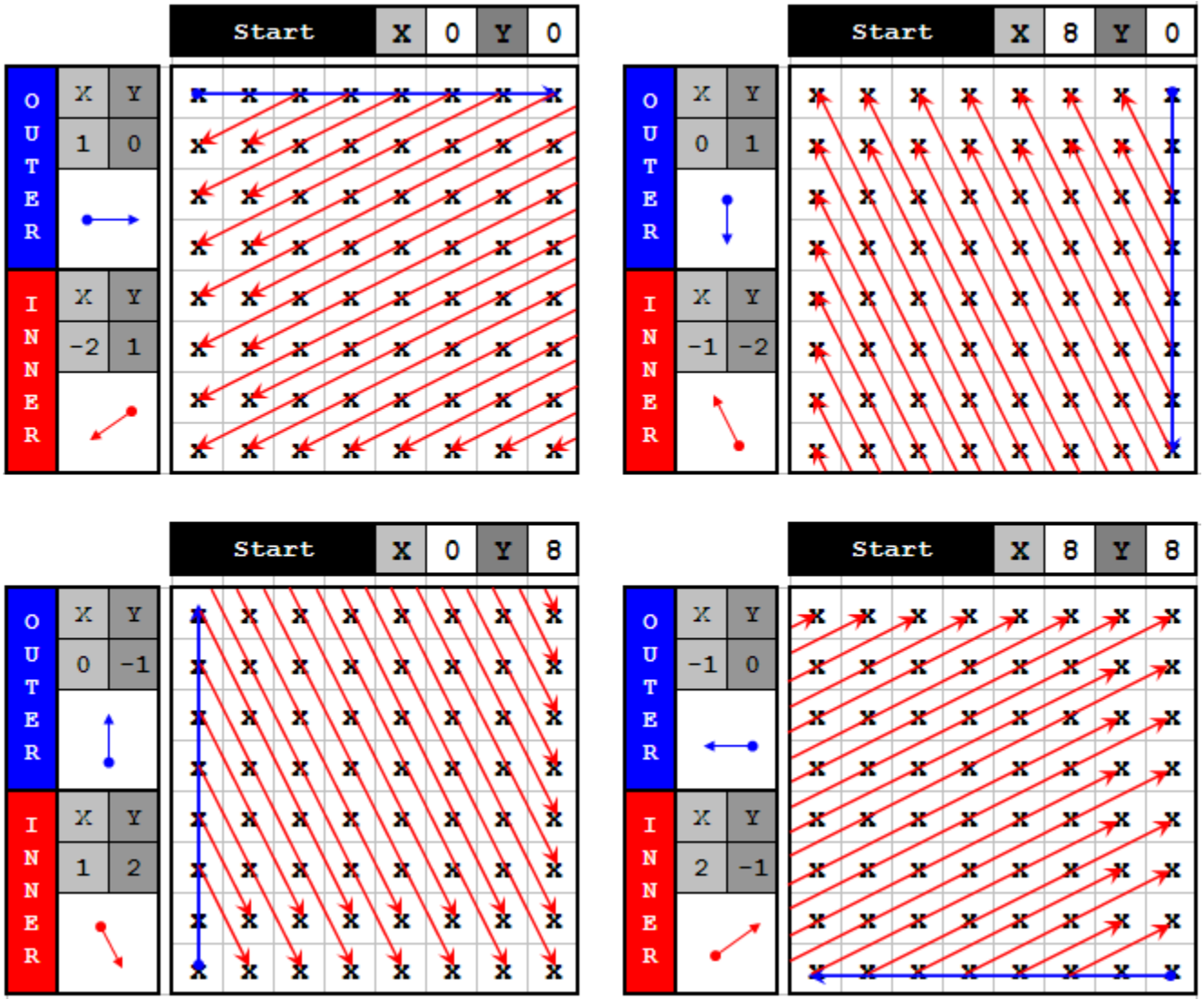
The local inner and outer loop xy-pair parameters alone can describe a large variety of primitive walking patterns. Below are 9 primitive walking patterns generated by varying only the inner and outer unit step vectors of the local loop:



- The top row shows the outer unit vector pointing down (+Y) and the inner unit vector pointing right (+X). Rows and columns can easily be skipped by increasing the unit step vectors above one.

- The middle row the outer unit vector pointing right (+X) and the inner unit vector pointing down (+Y). Again, rows and columns are skipped by increasing the unit step vectors beyond one.
- The last row shows the capability to walk angles not perpendicular to the edge. The 1st shows a 45° walking pattern by setting the inner unit vector to (-1,1). The 2nd shows a checkerboard pattern by skipping every other outer loop and retaining the inner unit vector of (-1,1). The 3rd shows a 26.5° walking pattern by setting the inner unit vector to (-2,1).

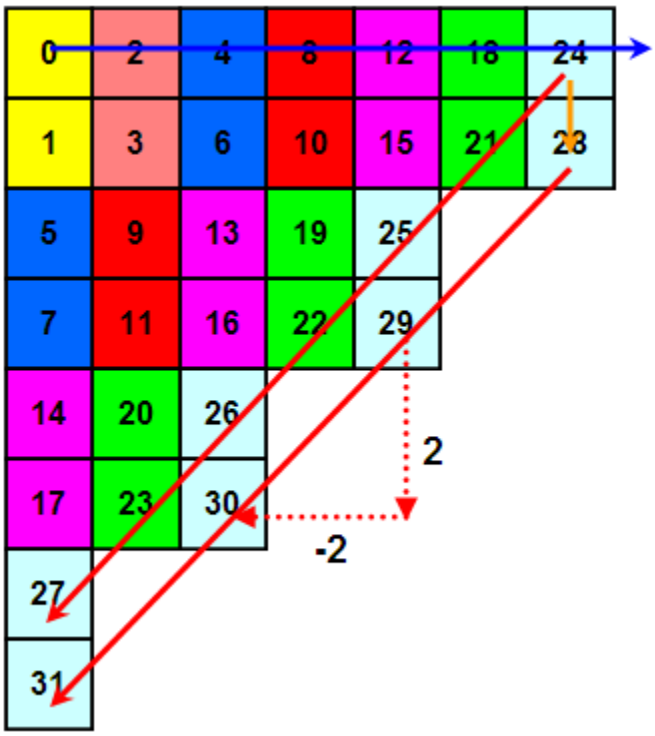
The block resolution, shown as [8,8], and the starting location, currently [0,0], can be varied and the above patterns can be stretched and rotated many ways. The diagram below shows an example of where the start position and unit step vectors can be set to achieve a full rotation of the same pattern:



MbAff-Like Special Case in Local Loop

The local loop has an additional middle loop that is used to achieve some specific walking patterns, with MBAFF mode especially in mind. A pattern to handle MBAFF AVC content is to walk the top macroblocks

of all macroblock pairs (MB-pairs) on a wavefront followed by the respective bottom macroblocks. The pattern is shown below.

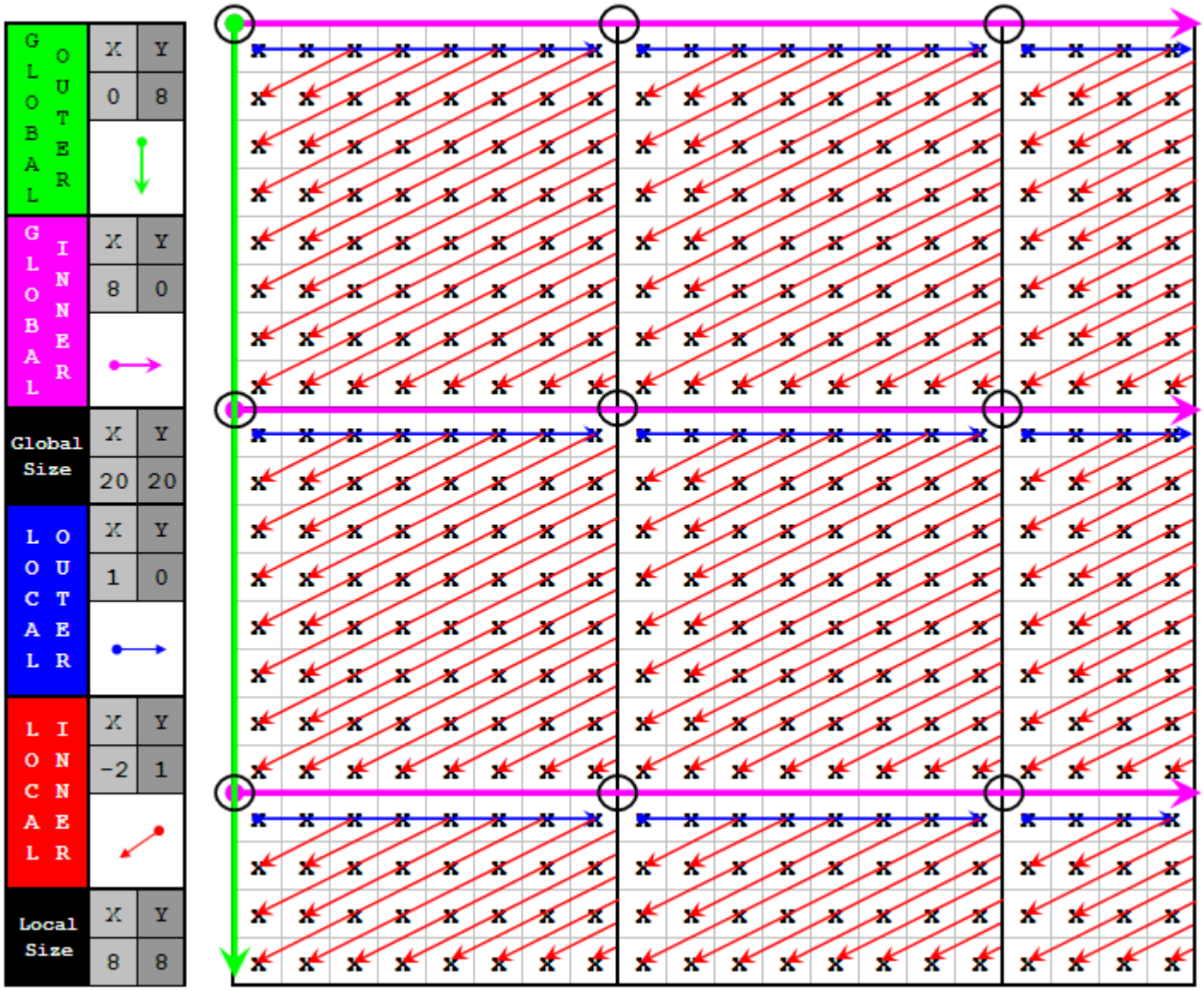


The outer loop unit step vector would be [1, 0] and the inner loop unit step vector would be [-2, 2]. A third loop is necessary to repeat the inner loop, only shifted down a unit before restarting. Thus, a middle loop with a unit step vector of [0,1] would achieve this MBAFF pattern. Additionally, the number of "extra steps" taken by the middle loop would be 1 in this case.

The addition of a middle loop also creates more overall flexibility, which seems necessary due to the integer-based unit step vector solution proposed (Manhattan distance issues etc.).

Global Loop

The same set of general parameters is used to describe the global loop as well. Thus, a global loop that is walking a raster-scan pattern can be combined with a local loop that is walking a 26.5° pattern (or vice-versa). As shown in the example below, if the local block size [8,8] is not an even multiple of the global resolution [20,20], the slack is still processed by dynamically changing the local block resolution.



The global loop will always resolve to be the upper-left corner of the local loop, shown above black circles. Note that local loop can still start in any corner of the local block, but the local (0,0) will always be the location where global loop begins the local loop, hence the upper-left corner.

The user can specify the starting location of the global loop as with the local loop. If the user were to set the global starting location to (16,16) in the previous example, after inverting the global outer and global inner unit step vectors the same pattern would be achieved in the reverse order. Note that the slack would still be handled along the right and bottom edge of the global image in that case. The user could have also started at (12,12) in which case the slack would be handled on the left and top faces.

Walker Algorithm Description

The walker algorithm has been tested and optimized in software. A high-level pseudo-code description is given below:

```
    Walker(){ //C-Style Pseudo-Code of Walker Algorithm        Load_Inputs_And_Initialize();
        While (Global_Outer_Loop_In_Bounds()){
            Global_Inner_Loop_Intialization();
            While (Global_Inner_Loop_In_Bounds()){
                Local_Block_Boundary_Adjustment();
                Local_Outer_Loop_Initialization();
                While (Local_Outer_Loop_In_Bounds()){
                    Local_Middle_Loop_Initialization();
                    While (Local_Middle_Steps_Remaining()){
                        Local_Inner_Loop_Initialization();
                        While (Local_Inner_Loop_Is_Shrinking()){
                            Execute();
                            Calculate_Next_Local_Inner_X_Y();
                        } //End Local Inner Loop
Calculate_Next_Local_Middle_X_Y();
                    } //End Local Middle Loop
Calculate_Next_Local_Outer_X_Y();
                    Calculate_Next_Local_Inverse_Outer_X_Y();
                } //End Local Outer Loop                Calculate_Next_Global_Inner_X_Y();
            } //End Global Inner Loop            Calculate_Next_Global_Outer_X_Y();
        } //End Global Outer Loop    } //End Walker
```

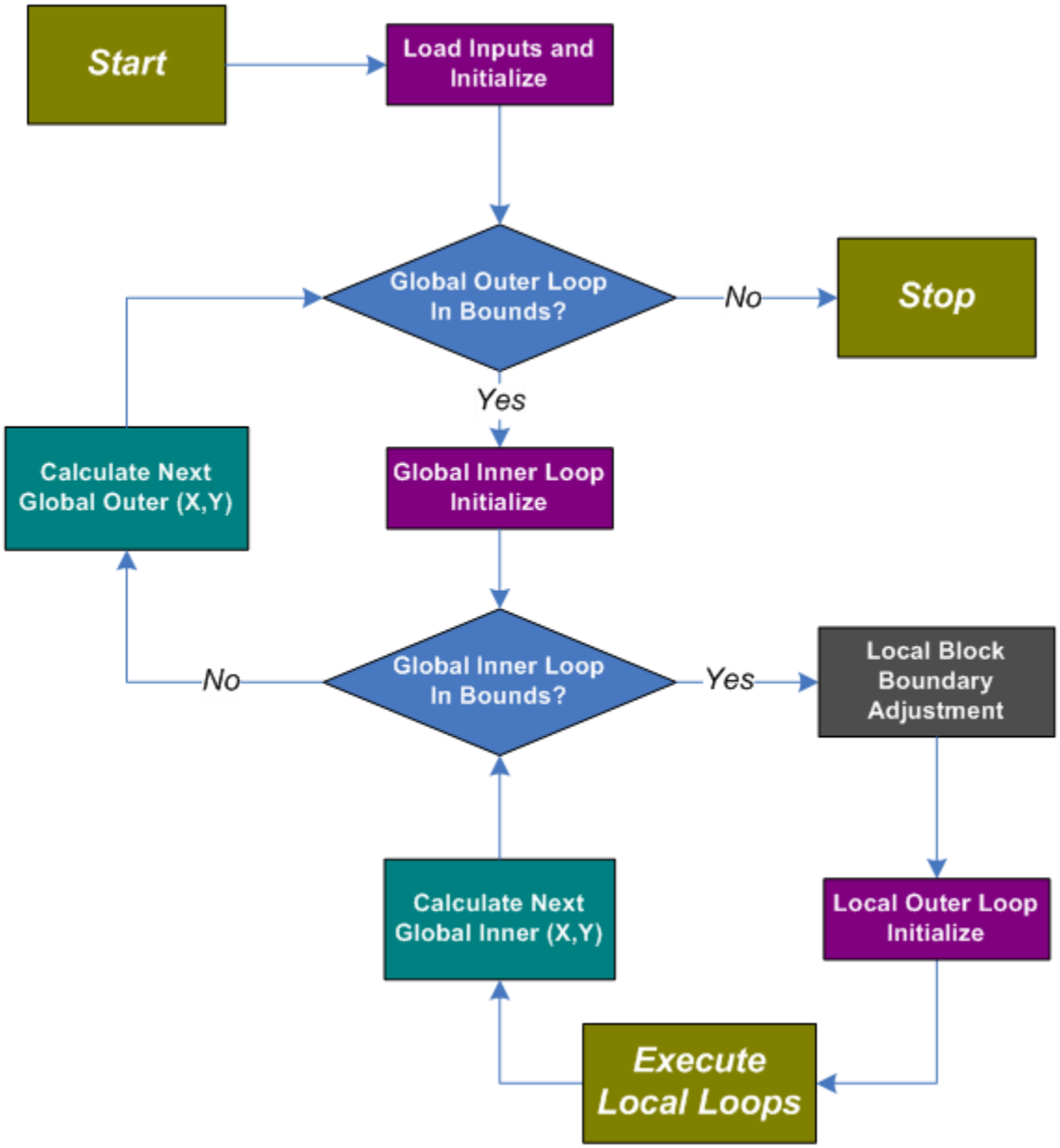
The pseudo-code has the following characteristics:

- There are 5 levels of iteration
- The highest 2 levels are called “global” and the lowest 3 levels are called “local”
  - The global loop is split into an outer and an inner loop.
  - The local loop is split into an outer, a middle, and an inner loop.
  - A bounding box for the global and local resolution is defined by the user.
  - The starting location within each bounding box is also specified by the user.
- Each of the 5 loops has its own persistent
  - Current position (x,y)
  - Unit step vector (x,y)
- The final output (x,y) is a summation of the global x,y and the local x,y.
- The next (x,y) for given level can be calculated while the next lower level is still executing. Additionally, the result can be used to check to see if the current level will execute again once control is returned.

The flow of the global outer and inner loops is:

1. Check a bound condition
2. Initialize the next level loop
3. Execute the next level loop
4. When the next level loop fails its condition, calculate the next position for the current loop level and repeat.

Walker algorithm flowchart for the Global Loop



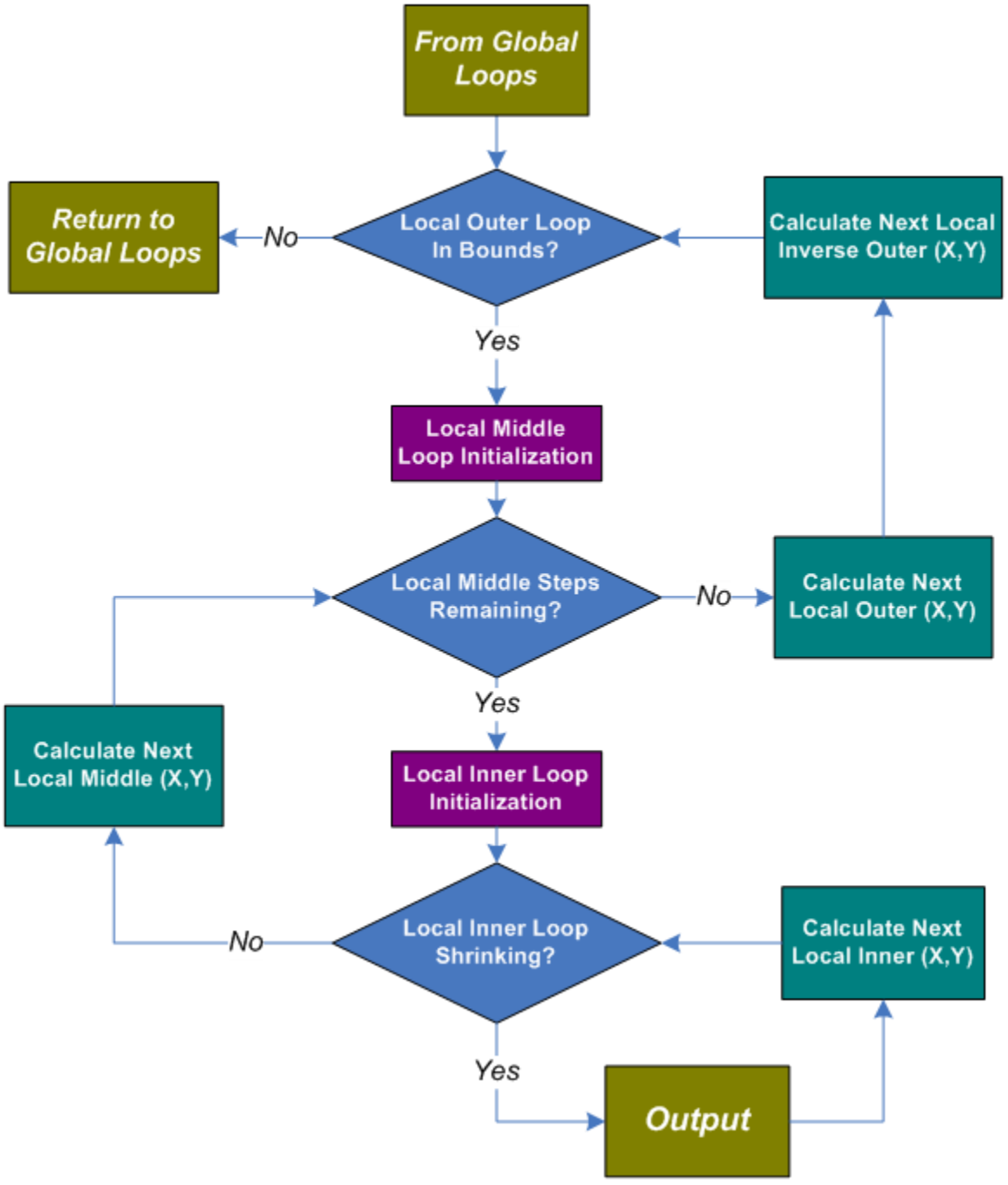
Take note of the grey box "Local Block Boundary Adjustment". This logic is necessary to adjust the local block size when the distance between the current global position to the edge of the image is less than the local resolution. Additionally, the local starting positions might be modified here as well if the defined starting position is larger than the new local block size.

The flow of the 3 local loops does not vary much from the 2 global loops. The differences are:

- In addition to a boundary check, the local middle loop also ensures the number of middle steps is less than or equal to the user defined "number of extra steps".

- The local inner loop only checks to see if the prior distance between the x,y starting and ending points are greater than their current distance. If this is true, it implies that the two inner loops are converging towards each other.
- When the middle loop check fails, both the starting points (local outer) and ending points (local inner) are updated.

Walker algorithm flowchart for the Local Loop



Barriers and Shared Local Memory

Barriers and Shared Local Memory can provide advantages to general media applications. Barriers can be used to synchronize between media threads more efficiently than using atomics, while SLM can be used to share data between tightly associated threads.

Allows Barriers and SLM to be used with a more generalized walker, as well as adding the ability to use the scoreboard at the same time. To implement this there needs to be an identifying number similar to the Thread Group ID which can be used to track and free resources.

For MEDIA_OBJECT_WALKER there are already numbers available which can be used as the group id – the various loop counts that the walker maintains. To provide flexibility, the programmer will be allowed to specify which loop counts will form the group id and which will count the threads inside each group. If cross-slice barriers are disabled (see section 1.7), the walker will be required to ensure that all threads in a group are dispatched to a single subslice so that the barrier and SLM are available to all group members. The programmer is responsible to ensure that the number of threads generated per group is not larger than the threads available in a subslice.

MEDIA_OBJECT_WALKER adds the X/Y values for the various loops added together to produce a single X/Y. Since walking patterns can be produced which have overlapping X/Y values, this can't be used for the global id, instead the execution counts (a count of how many times each loop is executed) are concatenated together to produce an id number. The GPGPU thread group id is a 96-bit number, so the 49 bits created by all the execution counts will fit easily.

The media walker has 5 nested loops for producing the X/Y; in addition there is an innermost color loop:

1. Color loop – 4 bits
2. Inner local loop execution count – 10 bits
3. Mid local loop execution count – 5 bits
4. Outer local loop execution count – 10 bits
5. Inner global loop execution count – 10 bits
6. Outer global loop execution count – 10 bits

The bits to use as the global id are specified with a parameter which specifies at which point to switch between the group id and the per thread id. Unused loops will always have execution counts of 0. The group id is formed from whichever execution counts are enabled in the Group ID Loop Select, with the selected execution counts concatenated into the LSBs of the group id, with any unused MSBs 0.

Example: if a MEDIA_OBJECT_WALKER specifies that the Outer local loop count and above will form the group id, then every iteration of the color, inner local, and mid local will dispatch a thread with the same group id. TSG will allocate a shared barrier and SLM (depending on what is specified in the Interface Descriptor) and ensure that all the threads go to the same subslice. The programmer ensures that this number of threads will fit on a single subslice, unless only global barriers are used, in which case the number of thread must fit on the system. The group id for this example is formed by cat(outer global exec count, inner global exec count, outer local exec count).

When the group id increments, the TSG will allocate a new barrier and SLM and pick an available subslice.

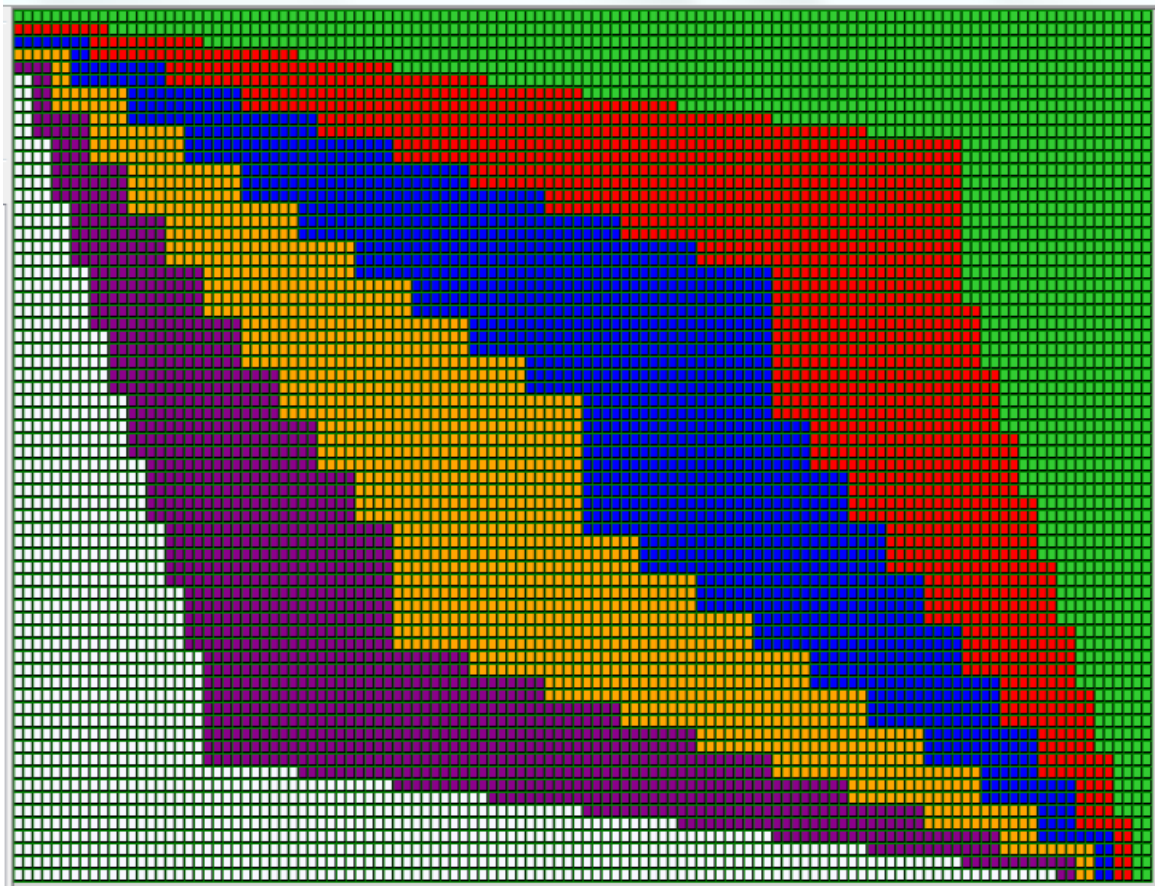
Before the MEDIA_OBJECT_WALKER command with groups there should be a MEDIA_STATE_FLUSH with a watermark bit and matching Interface Descriptor.

Flexible Dispatch of Local Loop

The local loop is automatically split between the availble subslices in such a way as to keep adjacent blocks next to each other to improve cache and execution efficency. The length of a single iteration of the local loop is split into equal segments, one for each subslice that is currently powered up.

The dispatches for the local loop are done such that one thread from each segment is dispatched before repeating the process: for example segment 0, thread 0 is followed by segment 1, thread 0 rather than segment 0, thread 1.

Example of splitting a wavefront between 6 destinations



Each color indicates a separate segment which is dispatched to a different subslice.

In the example above, each local loop walks a diagnal line from the lower left to the upper right, while the global loop steps between the lines. This is a typical usage model where the dependencies are to the left and above. The first few iterations are in the upper right corner and so have few blocks dispatched

per local loop. Farther down, the length of the local loop gets large enough that the 6 available subslices are full of threads from each segment running in parallel and being dispatched in an even manner.

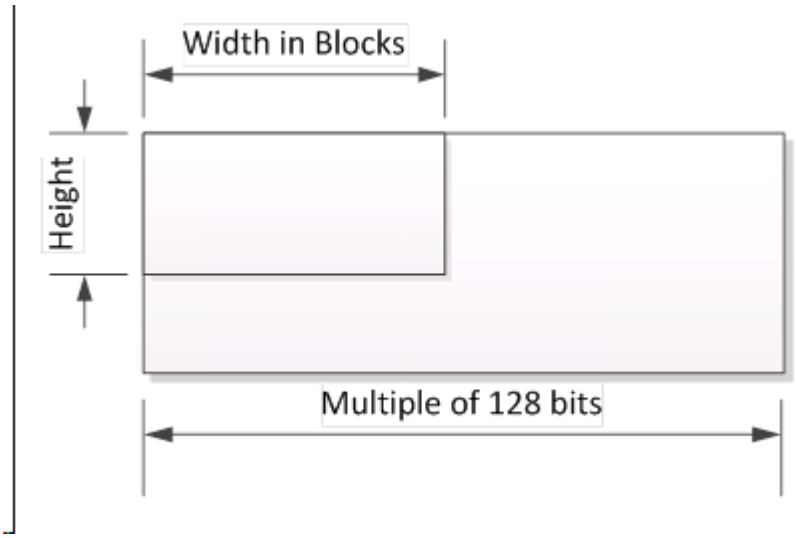
Masked Dispatch

Masked dispatch adds a dispatch mask to the Media_Object_Walker command. This provides a bit per dispatch which determines whether the dispatch is dropped or not. This allows Media_Object_Walker to be used with irregularly shaped areas. When using this mode, the walker should be programmed to walk the surface as a simple rectangle with X changing in the innermost loop.

The dispatch mask is stored in CURBE, which means that a Media_Object_Walker command that uses it cannot use CURBE as part of the dispatch payload. The CURBE is loaded with a bitmask with each bit corresponding to the X/Y blockid (not the pixel X/Y). The byte address of each bit is :

Byte_address = (Y * pitch + X) » 8

Pitch is the number of blocks in the surface horizontally rounded up to the next 128, 256 or 512 bits.



The Media_Object_Walker command allows a maximum 9-bit X and Y, which would correspond to a 512-bit wide x 512 deep surface in CURBE. For 16x16 pixel blocks this would be 8k x 8k pixels.

Note that in this mode the walker will dispatch in sequence rather than splitting the local loop into segments which are dispatched to individual sub-slices (as described in “Flexible Dispatch of Local Loop”.

Scoreboard Control

A hardware mechanism controls the dispatch of root threads. Without using this hardware mechanism, only the dispatch of a SRT is managed by a parent root thread using the SRT message to TS.

There is a scoreboard hardware in TS unit. The scoreboard is addressed by the 18-bit (X, Y) scoreboard field in VFE DWord, where (X, Y) is typically used as the Cartesian coordinate of the working unit in a 2D frame but may be interpolated in other ways. When a root thread is dispatched, the entry at (X, Y) is marked. When the root thread is terminated, the corresponding bit in the scoreboard is cleared.

Each root thread may have up to eight dependencies. The dependency relation is described by the state value of Scoreboard Controls in terms of related distance of (deltaX, deltaY). There is a global scoreboard enabling in the state as well as the-per thread enabling for each dependency.

TS stalls the dispatch of a root thread if any scoreboard entry, which is denoted by (Scoreboard X + deltaX, Scoreboard Y + deltaY), matching with any enabled dependencies is marked as in-flight. The thread is dispatched only after all dependencies are cleared.

For a root thread, TS stalls the dispatch of the thread only if the dependent scoreboard entries of the thread are marked. It does not automatically stalls the dispatch for destination collision if (deltaX = 0, deltaY=0) is not set in the scoreboard state. This kind of scoreboard destination collision is due to the scoreboard wrap-around (or aliasing), which must be avoided. With 9-bit per X, Y field, the hardware scoreboard can support a frame that is subdivided up to 512x512 threads without a scoreboard aliasing.

In addition to the above 'stalling scoreboard', Media Pipe may also support a non-stalling scoreboard. With non-stalling, a thread is dispatched with the dependent threads marked. The thread dependency affects the issuing of a *sendc* instruction. See vol5d Execution Unit ISA for details.

Scoreboard Support in Device Hardware

Device	Stalling scoreboard	Non-Stalling scoreboard
	Yes	Yes

Programming Note	
Context:	Scoreboard Control
• The hardware scoreboard only handles root threads, but not child threads. This limitation may be revisited when future application requirement changes. • The usage of hardware scoreboard and SRT are mutually exclusive. In other words, when hardware scoreboard is used, SRT should not be issued.	

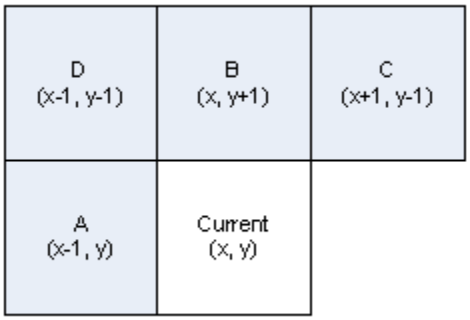
AVC-Style Dependency Example

For AVC decoding, dependencies for a given macroblock may be set based on the availability of neighbor macroblocks, namely A, B, C, D and left-bottom neighbors (left-bottom only if MbAff = 1), as well as the current macroblock's address, MbAff flag and FieldMbFlag. For a macroblock in a progressive frame picture or a field picture, one macroblock may depend on up to four neighbors, A, B, C and D as shown in Neighbor addresses of a macroblock in a progressive frame picture (MbAff = 0) or a field picture with up to 4 dependencies. For a macroblock in a MbAff pair, it may depend on up to three, five or eight neighbors as shown in Neighbor addresses of the first macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies and Neighbor addresses of the second macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies, based on the current macroblock's address and FieldMbFlag.

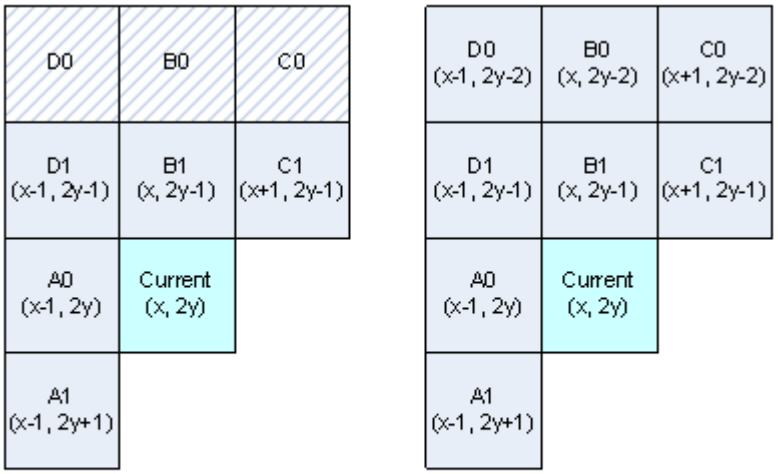
The neighbor's availability depends on the corresponding **IntraPredAvailFlagA|B|C|D|E** flags for the macroblock (or the macroblock pair). Hardware assumes that the flags are set correctly in the

MEDIA_OBJECT_EX command as shown in Macroblock indices for field picture destination. For simplicity, the left neighbor pair (A0 and A1) availability for a MbAff macroblock can be determined as a group by **IntraPredAvailFlagA | IntraPredAvailFlagE**. For the second macroblock in a 'frame' MbAff pair, it depends on the first macroblock in the pair and it is always available.

Neighbor addresses of a macroblock in a progressive frame picture (MbAff = 0) or a field picture with up to 4 dependencies



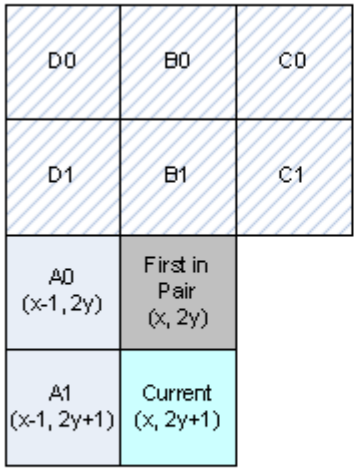
Neighbor addresses of the first macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies



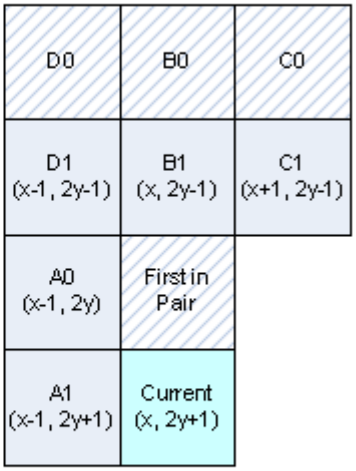
(a) Neighbors for the first macroblock in a 'frame' MbAff pair

(b) Neighbors for the first macroblock in a 'field' MbAff pair

Neighbor addresses of the second macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies



(b) Neighbors for the second macroblock in a 'frame' MbAff pair



(b) Neighbors for the second macroblock in a 'field' MbAff pair

Neighbor Availability

MbAff	FieldMbFlag	VertOrigin[0]	A	B	C	D	LB	Description
0	0/1	0/1	□	□	□	□	•	Progressive or Field picture
1	0	0	□	□	□	□	□	1st Frame MbAff macroblock
1	0	1	□	na	0	na	□	2nd Frame MbAff macroblock
1	1	0	□	□	□	□	□	1st Field MbAff macroblock
1	1	1	□	□	□	□	□	2nd Field MbAff macroblock

Interface Descriptor Selection

In VLD mode, the Interface Descriptor Offset field in the MEDIA_OBJECT command is ignored by hardware. Instead, the interface descriptor offset is computed by hardware based on the decoded macroblock parameters and a remapping table.

First a macroblock index is computed based on parameters such as picture structure, motion type, prediction type, DCT type, intra-coding type and motion vector present information. *Interface Descriptor Selection* provides the macroblock index table for a frame-picture destination buffer (with Picture Structure = 11). *Interface Descriptor Selection* shows macroblock indices for a field-picture destination

buffer (with Picture Structure = 01 or 10). As Picture Structure is a state variable that is not changed until a pipeline flush, the macroblock indices can be computed separately for different Picture Structure.

After the macroblock index is computed, it is used as the index into the Interface Descriptor Remap Table to derive the final interface descriptor offset value. The Interface Descriptor Remap Table is provided as part of the VLD state.

The interface descriptor offset value multiplied by the interface descriptor size is then added to the interface descriptor base pointer to generate the interface descriptor pointer for the post-VLD thread.

The last three columns in *Interface Descriptor Selection* and *Interface Descriptor Selection* indicate whether a macroblock index is applicable for a given Picture Coding type (I, P or B). A 'Y' (or a 'N') means the macroblock index on the row is valid (or invalid) for the Picture Type shown on the column. Taking a frame picture destination for example, only macroblock indices 0 and 8 are valid for an I-picture; indices 0-3 and 8-11 are valid for a P-picture; and for a B-picture, only indices 3 and 11 are not valid.

Developers can use the remap table for kernel development to fine-tune system performance and reduce software complexity. For example, if the destination is a frame picture, the kernel for a macroblock with dual-prime motion in a P-picture (macroblock index = 3) may be identical to that for a macroblock with bidirection field motion in a B picture (macroblock index = 7). A common set of interface descriptors can be configured once for frame picture destinations, and reused without change when the destination is of I-, P- and B- picture coding type.

In another case, if it is determined that kernel software is responsible for handling DCT types for a frame picture destination, then macroblock index i and i+8 , for i = 0 to 7, can be mapped to the same interface descriptor.

Macroblock Indices for Frame Picture Destination

Macroblock Index	Interface Descriptor Kernel Function (Frame Picture Destination)	I	P	B
0	I macroblock	Y	Y	Y
1	Forward frame motion	N	Y	Y
2	Forward field motion	N	Y	Y
3	P picture, dual-prime motion	N	Y	N
4	Backward frame motion	N	N	Y
5	Backward field motion	N	N	Y
6	Bidirectional frame motion	N	N	Y
7	Bidirectional field motion	N	N	Y
8	I macroblock with field DCT	Y	Y	Y
9	Forward frame motion with field DCT	N	Y	Y
10	Forward field motion with field DCT	N	Y	Y
11	P picture, dual-prime motion with field DCT	N	Y	N
12	Backward frame motion with field DCT	N	N	Y

Macroblock Index	Interface Descriptor Kernel Function (Frame Picture Destination)	I	P	B
13	Backward field motion with field DCT	N	N	Y
14	Bidirectional frame motion with field DCT	N	N	Y
15	Bidirectional field motion with field DCT	N	N	Y

Macroblock Indices for Field Picture Destination

Macroblock Index	Interface Descriptor Kernel Function (Field Picture Destination)	I	P	B
0	I macroblock	Y	Y	Y
1	Forward field motion	N	Y	Y
2	Forward 16x8 motion	N	Y	Y
3	P picture, dual-prime motion	N	Y	N
4	Backward field motion	N	N	Y
5	Backward 16x8 motion	N	N	Y
6	Bidirectional field motion	N	N	Y
7	Bidirectional 16x8 motion	N	N	Y

VC1-Style Dependency Example

For VC1, only one dependency may be set depending on the availability of the upper neighbor macroblock.

Macroblock sequence order in a VC-1 picture with WidthInMblk = 5 and HeightInMblk = 6

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24
5	25	26	27	28	29

Multiple Slice Considerations

For products with multiple slices, the Render Cache is separate per slice with no hardware coherency. This means that the programmer must ensure coherency by one of these methods:

- Using write commit when writing to the Render Cache.
- Using Data Cache instead of the Render Cache.

- Different slices only access separate cache lines. using a hashing algorithm combined with the slice select bits of the MEDIA_OBJECT/GPGPU_OBJECT commands.

Interrupt Latency

Command Streamer is capable of context switching between primitive commands.

For all independent threads, it is not much a problem. The interrupt latency is dictated by the longest command that is likely to have the largest number of threads. For VLD mode, such a command may be corresponding to a largest slice in a high definition video frame. This is application dependent, there are not much host software can do. For Generic mode, programmer should consider to constrain the compute workload size of each thread.

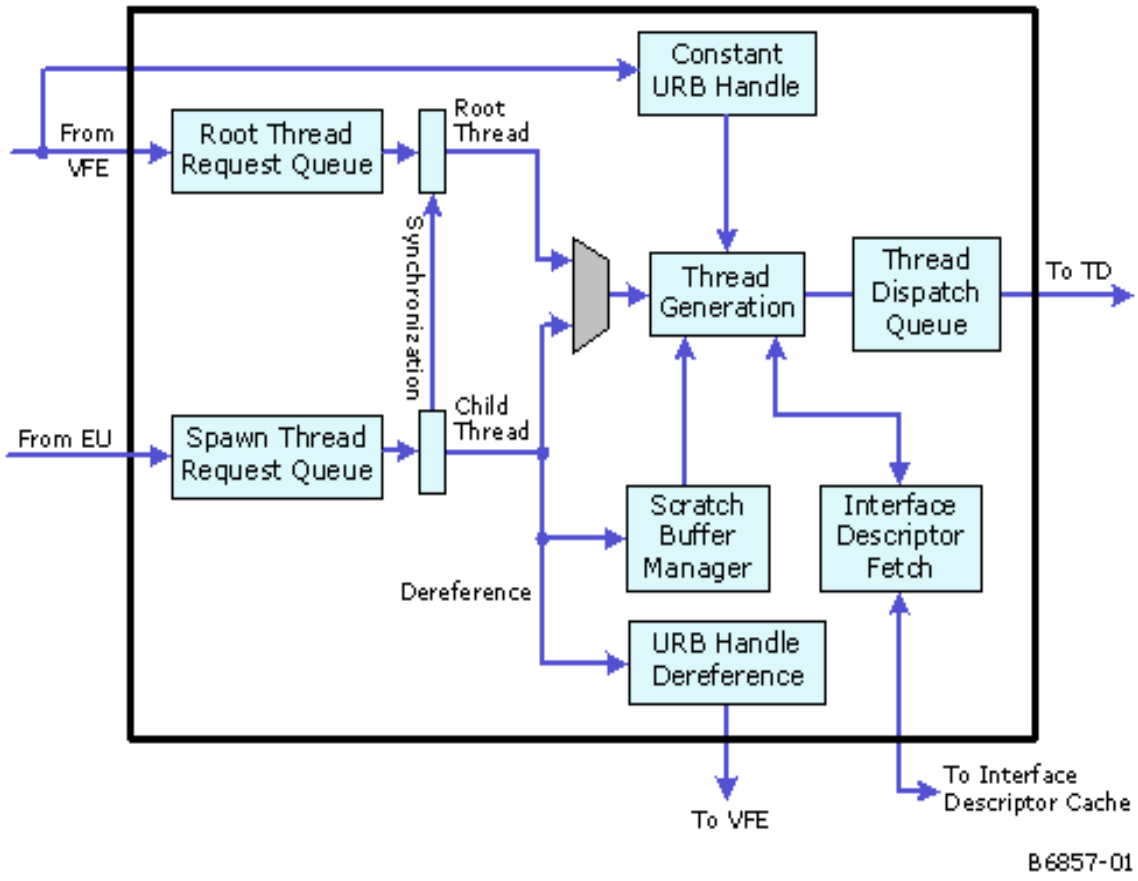
In modes with child threads, a root thread may persist in the system for long period of time – staying until its child threads are all created and terminated. Therefore, the corresponding primitive command may also last for long time. The Software designer should partition the workload to restrict the duration of each root thread. For example, this may be achieved by partitioning a video frame and assigning separate primitive commands for different data partitions.

In modes with synchronized root threads, a synchronized root thread is dependent on a previous root or child thread. This means context switch is not allowed between the primitive command for the synchronized root thread and the one for the depending thread. So no command queue arbitration should be allowed between them. Software designer should also restrict the duration of such non-interruptible primitive command segments.

Thread Spawner Unit

The Thread Spawner (TS) unit is responsible for making thread requests (root and child) to the Thread Dispatcher, managing scratch memory, maintaining outstanding root thread counts, and monitoring the termination of threads.

Thread Spawner block diagram



Root Threads and Child Threads

Thread requests sourced from VFE are called **root threads**. These threads may be creating subsequent child threads.

Root Threads

A root thread may be a macroblock thread created by VFE as in VLD mode, or may be a general-purpose thread assembled by VFE according to full description provided by host software in Generic mode. Thread requests are stored in the Root Thread Queue. TS keeps everything needed to get the root threads ready for dispatch and then tracks dispatched threads until their retirement.

TS arbitrates between root thread and child thread. The root thread request queue is in the arbitration only if the number of outstanding threads does not exceed the maximum root thread state variable. Otherwise, the root thread request queue is stalled until some other root threads retire/terminate.

Once a root thread is selected to be dispatched, its lifecycle can be described by the following steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.
    - Once TS receives the interface descriptor, it checks whether maximum concurrent root thread number has reached to determine whether to make a thread dispatch request or to stall the request until some other root threads retire. If the thread requests the use of scratch memory, it also generates a pointer into the scratch space.
2. TS then builds the transparent header and the R0 header.
3. Finally, TS makes a thread request to the Thread Dispatcher.
4. TS keeps track of dispatched thread, and monitors messages from the thread (resource dereference and/or thread termination). When it receives a root thread termination message, it can recover the scratch space and thread slot allocated to it. The URB handle may also be dereferenced for a terminated root thread for future reuse. It should be noted that URB handle dereference may occur before a root thread terminates. See detailed description in the Media Message section.
    - It is the root thread's responsibility (software) to guarantee that all its children have retired before the root thread can retire.

URB Handles

VFE is in charge of allocating URB handles for root threads. One URB handle is assigned to each root thread. The handle is used for the payload into the root thread.

Children Present is a command variable in the _OBJECT command.

If Children Present is not set (root-without-child case), TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched.

If Children Present is set (root-with-child case), the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread.

Root to Child Responsibilities

Any thread created by another thread running in an EU is called a **child thread**. Child threads can create additional threads, all under the tree of a root which was requested via the VFE path.

A root thread is responsible of managing pre-allocated resources such as URB space and scratch space for its direct and indirect child threads. For example, a root thread may split its URB space into sections. It can use one section for delivering payload to one child thread as well as forwarding the section to the child thread to be used as return URB space. The child thread may further subdivide the URB section into subsections and use these subsections for its own child threads. Such process may be iterated. Similarly, a root thread may split its scratch memory space into sections and give one scratch section for one child thread.

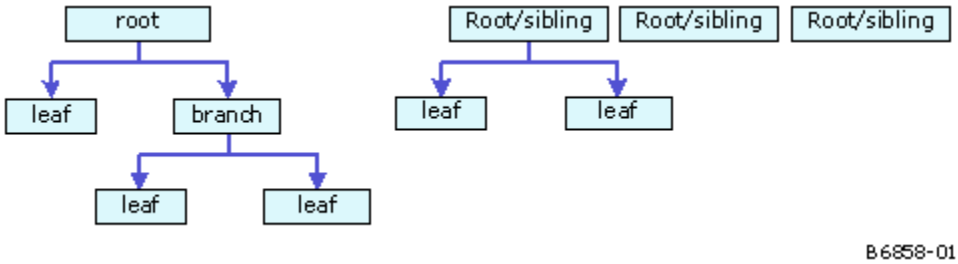
TS unit only enforces limitation on number of outstanding root threads. It is the root threads' responsibility to limit the number of child threads in their respected trees to balance performance and avoid deadlock.

Multiple Simultaneous Roots

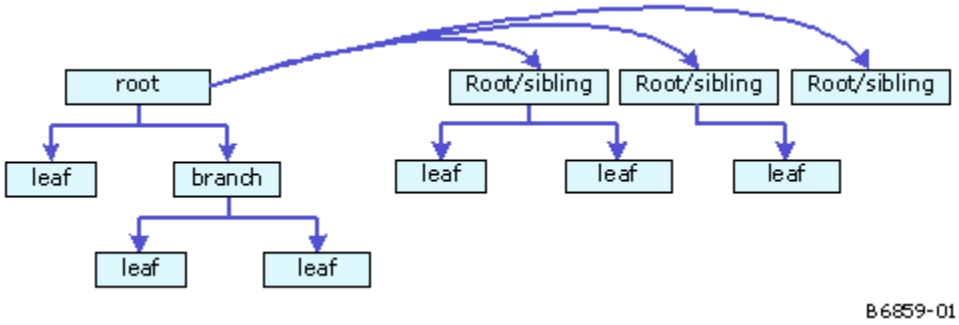
Multiple root threads are allowed concurrently running in GEN4 execution units. As there is only one scratch space state variable shared for all root threads, all concurrent root thread requiring scratch space share the same scratch memory size. *Multiple Simultaneous Roots* depicts two examples of thread-thread relationship. The left graph shows one single tree structure. This tree starts with a single root thread that generates many child threads. Some child threads may create subsequent child threads. The right graph shows a case with multiple disconnected trees. It has multiple root threads, showing sibling roots of disconnected trees. Some roots may have child threads (branches and leafs) and some may not.

There is another case (as shown in *Multiple Simultaneous Roots*) where multiple trees may be connected. If a root is a synchronized root thread, it may be dependent on a preceding sibling root thread or on a child thread.

Examples of thread relationship



A example of thread relationship with root sibling dependency



Synchronized Root Threads

A synchronized root thread (SRT) originates from a MEDIA_OBJECT command with Thread Synchronization field set. Synchronized root threads share the same root thread request queue with the non-synchronized roots. A SRT is not automatically dispatched. Instead, it stays in the root thread request queue until a spawn-root message is at the head of the child thread request queue. Conversely, a spawn-root message in the child thread request queue will block the child thread request queue until the head of root thread request queue is a SRT. When they are both at the head of queues, they are taken out from the queue at the same time.

A spawn-root message may be issued by a root thread or a child thread. There is no restriction. However, the number of spawn-root messages and the number of SRT must be identical between state changes. Otherwise, there can be a deadlock. Furthermore, as both requests are blocking, synchronized root threads must be used carefully to avoid deadlock.

When Scoreboard Control is enabled, the dispatch of a SRT originated from a MEDIA_OBJECT_EX command is still managed by the same way in addition to the hardware scoreboard control.

Deadlock Prevention

Root threads must control deadlock within their own child set. Each root is given a set of preallocated URB space; to prevent deadlock it must make sure that all the URB space is not allocated to intermediate children who must create more children before they can exit.

There are limits to the number of concurrent threads. The upper bound is determined by the number of execution units and the number of threads per EU. The actual upper bound on number of concurrent threads may be smaller if the GRF requirement is large. Deadlock may occur if a root or intermediate parent cannot exit until it has started its children but there is no space (for example, available thread slot in execution units) for its children to start.

To prevent deadlock, the maximum number of root threads is provided in VFE state. The Thread Spawner keeps track of how many roots have been spawned and prevents new roots if the maximum has been reached. When child threads are present, it is software's responsibility to constrain child thread generation, particularly the generation of child threads that may also spawn more child threads.

Child thread dispatch queue in TS is another resource that needs to be considered in preventing deadlock. The child thread dispatch queue in TS is used for (1) message to spawn a child thread, (2) message to spawn a synchronized root thread, and (3) thread termination message. If this queue is full, it will prevent any thread to terminate, causing deadlock.

For example, if an application only has one root thread (max # of root threads is programmed to be one). This root thread spawns child threads. In order to avoid deadlock, the maximum number of outstanding child thread that this root thread can spawn is the sum of the maximum available thread slots plus the depth of the child thread dispatch queue minus one.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1)

Adding other root threads (synchronized and/or non-synchronized) to the above example, the situation is more complicated. A conservative measure may have to use to prevent deadlock. For example, the root thread spawning child threads may have to exclude the max number of root threads as in the following equation to compute the maximum number of outstanding child threads to be dispatched.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1) – (Max Root Threads-1)

Child Thread Life Cycle

When a (parent) thread creates a child thread, the parent thread behaves like a fixed function. It provides all necessary information to start the child thread, by assembling the payload in URB (including R0 header) and then sending a spawn thread message to TS with following data:

- An interface descriptor pointer for the child thread.
- A pointer for URB data

The interface descriptor for a child may be different from the parent – how the parent determines the child interface descriptor is up to the parent, but it must be one from the interface descriptor array on the same interface descriptor base address.

The URB pointer is not the same as a URB handle. It does not have an URB handle number and does not appear in any handle table. This is acceptable because the URB space is never reclaimed by TS after a child is dispatched, but rather when the parent releases its original handles and/or retires.

Programming Note	
Context:	Child Thread Life Cycle
The child request is stored in the child thread queue. The depth of the queue is limited to 8, overrun is prevented by the message bus arbiter which controls the message bus. The arbiter knows the depth of the queue and will only allow 8 requests to be outstanding until the TS signals an entry has been removed.	

As mentioned previously, child threads have higher priority over root threads. Once TS selects a child thread to dispatch, it follows these steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.
2. TS then builds the transparent header but not the R0 header.
3. Finally, TS makes a thread request to the Thread Dispatcher.
4. Once the dispatch is done, TS can forget the child – unlike roots, no bookkeeping is done that has to be updated when the child retires.

If more data needs to be transferred between a parent thread and its child thread than that can fit in a single URB payload, extra data must be communicated via shared memory through data port.

Arbitration between Root and Child Threads

When both root thread queue and child thread queue are both non-empty, TS serves the child thread queue. In other words, child threads have higher priority over root threads. The only condition that the child thread queue is stalled by the root thread queue is that the head of child thread queue is a root-synchronization message and the head of root thread queue is not a synchronized root thread.

Persistent Root Thread

A persistent root thread in general stays in the system for a long period of time. It is normally a parent thread, and only one PRT is allowed in the system at a time.

On a context switch interrupt, instead of proceeding to completion, a PRT can save its software context and terminate. The PRT can be restarted later, even if it had completed normally the last time it was executed. Therefore, the PRT must always save enough context (via data port messages to a predefined surface) to allow it to restart from where it left off (including determining that it has nothing left to do).

Because only one PRT can execute at a time, once the next PRT starts, the previous one will never be restarted, thus the context save surface can be reused from one PRT to the next.

A PRT may check the Thread Restart Enable bit in the R0 header to find out whether it is a fresh start or resumed from a previous interrupt and then can continue operations from that previously saved context.

A PRT can be interleaved with other root (such as parent root thread, or synchronized root thread) and child threads. A parent root thread is not necessarily a PRT, and doesn't have to be as long as it can be finished in deterministic time that is shorter than required for fine-grain context switch interrupt.

Use of PRT must follow the following rule:

- There can only be one PRT in the media pipeline at a given time. That means, there shall not be any other media primitive commands (MEDIA_OBJECT or MEDIA_OBJECT_EX) between it and the previous MI_FLUSH command. In other words, when multiple such PRTs are used in a sequence of media primitive commands, MI_FLUSH must be inserted.

Media State Model

The media state model is based on in-line state load mechanism. VFE state, URB configuration and Interface Descriptors are loaded to VFE hardware through state commands.

All Interface Descriptors have the same size and are organized as a contiguous array in memory. They can be selected by Interface Descriptor Index for a given kernel. This allows different kinds of kernels to coexist in the system.

Pipeline (Media) Bits[28:27]	Opcode Bits[26:24]	Sub Opcode Bits[23:16]	Command
2h	0h	00h	MEDIA_VFE_STATE
2h	0h	01h	MEDIA_CURBE_LOAD
2h	0h	02h	MEDIA_INTERFACE_DESCRIPTOR_LOAD

Media State and Primitive Commands

This section contains various commands for media, all with the RenderCS source.

MEDIA_VFE_STATE

MEDIA_CURBE_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Interface Descriptor Data payload as pointed to by the Interface Descriptor Data Start Address:

INTERFACE_DESCRIPTOR_DATA

Programming Restriction: Back to back interface descriptor load commands, must have a VFE State command inserted between them.

MEDIA_STATE_FLUSH

The MEDIA_OBJECT command is the basic media primitive command for the media pipeline. It supports loading of inline data as well as indirect data. At least one form of payload (either inline, indirect, or CURBE) must be sent with the MEDIA_OBJECT.

MEDIA_OBJECT

MEDIA_OBJECT_PRT

MEDIA_OBJECT_GRPID

The MEDIA_OBJECT_WALKER command uses the hardware walker in VFE for generating threads associated with a rectangular shaped object. It only supports loading of inline data or CURBE but not indirect data. At least one form of payload must be sent. Control of scoreboards (up to 8) is implicit based on the (X, Y) address of the generated thread and the scoreboard control state.

The command can be used only in Generic modes.

When **Use Scoreboard** field is set, the (X, Y) address and the Color field of the generated thread are used in the hardware scoreboard and the thread dependencies are set by states from the MEDIA_VFE_STATE command.

One or more threads may be generated by this command. This command does not support indirect object load. When inline data is present, it is repeated for all threads it generates. Unlike CURBE, which requires pipeline flush for change, continued change of this kind of 'global' (in the sense of shared by multiple threads from this command) data is supported when MEDIA_OBJECT_WALKER commands are issued without a pipeline flush in between.

MEDIA_OBJECT_WALKER

Media State and Primitive Command Workarounds

Media State and Primitive Commands have some subtle programming restrictions and workarounds, as listed below.

Programming Note
The MEDIA_STATE_FLUSH command is updated to optionally specify all the resources required for the next thread group via an interface descriptor – if the resources are not available the group cannot start.
When a context switches between using the Stalling Scoreboard and the Non-stalling Scoreboard with the Scoreboard Type field in MEDIA_VFE_STATE a stalling PIPE_CONTROL must be placed before the MEDIA_VFE_STATE command.

Media Payload

There are 3 types of payload that the media kernels can have, but not all of them are allowed. The following table lists the legal combinations:

Workload	Commands	Data Stored
Media(Legacy)	Media_Object	CURBE
	Media_Object	INDIRECT
	Media_Object	INLINE
	Media_Object	CURBE+INLINE
	Media_Object	CURBE+INDIRECT
	Media_Object	INLINE+INDIRECT
	Media_Object	CURBE+ INLINE+INDIRECT
	Media_Object_Walker	CURBE
	Media_Object_Walker	INLINE
	Media_Object_Walker	CURBE+INLINE
Media using Barrier/SLM	Media_Object_GRPID	CURBE
	Media_Object_GRPID	INDIRECT
	Media_Object_GRPID	INLINE
	Media_Object_Walker (with group id)	CURBE

Workload	Commands	Data Stored
	Media_Object_Walker (with group id)	INLINE

For legacy media the payload is simple - each media_object or each dispatch in media_object_walker gets all the payload specified.

For media using barriers/SLM the CURBE and INDIRECT payload is similar to the GPGPU payload method where the payload is split into per-thread and cross-thread sections and each thread in a group gets a the cross-thread and a unique section of per-thread payload. The payload is only split between the threads in a group, and each group gets a duplicate version of the payload.

Media Messages

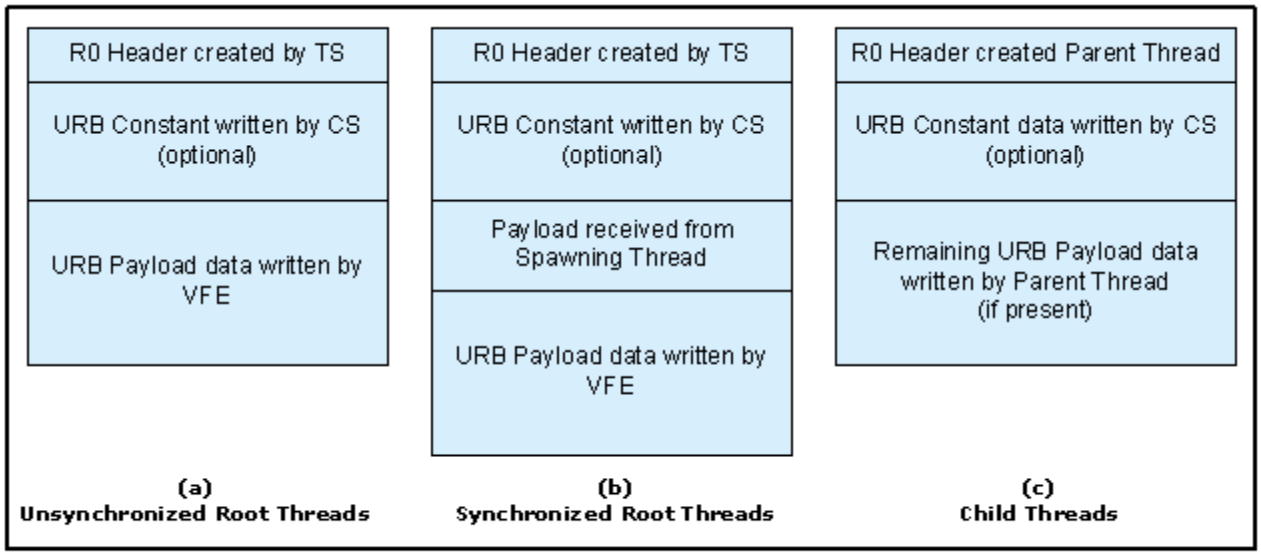
All message formats are given in terms of dwords (32 bits) using the following conventions:

- Dispatch Messages: **R**p.d
- SEND Instruction Messages: **M**p.d

Thread Payload Messages

The root thread's register contents differ from that of child threads, as shown in *Thread Payload Messages*. The register contents for a synchronized root thread (also referred to as 'spawned root thread') and an unsynchronized one are also different. Whether the URB Constant data field is present or not is determined by the interface descriptor of a given thread. This applies to both root and child threads. When URB Constant data field is present for a synchronized root thread, URB constant data field is before the data field received from the spawning thread, which is also before the URB payload data.

Thread payload message formats for root and child threads



Generic Mode Root Thread

The following table shows the R0 register contents for a Generic mode root thread, which is generated by TS. The remaining payloads are application dependent.

R0 Header of a Generic Mode Root Thread

DWord	Bits	Description
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
R0.4	31:5	**Binding Table Pointer.** The 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** The amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2	31:28	Reserved: MBZ
	27:16	Reserved: MBZ
	15:10	Reserved: MBZ
	9:4	**Interface Descriptor Offset.** The offset from the interface descriptor base pointer to the interface descriptor that applies to this object, in units of interface descriptors. Format = U6
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX): This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
R0.1	31:28	Reserved: MBZ
	26:16	**Scoreboard Y** This field provides the Y term of the scoreboard value of the current thread. Format = U11
	15:11	Reserved: MBZ
	10:0	**Scoreboard X** This field provides the X term of the scoreboard value of the current thread. Format = U11
R0.0	31:24	Reserved: MBZ

DWord	Bits	Description
	23:16	**Scoreboard Mask.** Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. **Bit n (for n = 0...7):** Scoreboard n is dependent, where bit 24 maps to n = 0. Format = TRUE/FALSE
	15:0	**URB Handle.** This is the URB handle indicating the URB space for use by the root thread and its children.

Root Thread from MEDIA_OBJECT_PRT

The root thread payload message for a MEDIA_OBJECT_PRT command has a fixed format independent of the VFE mode (e.g. Generic mode or AVC-IT mode). One example GRF register location is given for the condition that CURBE is disabled.

Root Thread Payload Layout for a MEDIA_OBJECT_PRT Command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	N/A	**Constants from CURBE when CURBE is enabled** m is a non-negative value.
R(m+1)	R1	**In-line Data block**

The R0 header field is as the following, which is the same as in other modes except the Thread Restart Enable bit (bit 0 of R0.2).

R0 Header of the Thread Payload of a MEDIA_OBJECT_PRT Command

DWord	Bit	Description
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1K-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9:8	Reserved: MBZ
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space.

DWord	Bit	Description
		Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2	31:4	**Interface Descriptor Pointer.** Specifies the 16-byte aligned pointer to *this thread's* interface descriptor. Can be used as a base from which to offset child thread's interface descriptor pointers. Format = GeneralStateOffset[31:4]
	3:1	Reserved: MBZ
	0	**Thread Restart Enable.** If set, indicates that the persistent root thread (PRT) is being restarted, and context should be restored from the context save area before executing. Format = Enable
R0.1	31:0	Reserved: MBZ
R0.0	31:16	Reserved: MBZ
	15:0	**URB Handle.** This is the URB handle indicating the URB space for use by the root thread and its children.

The inline data block field is the same as in the MEDIA_OBJECT_EX command with zero-filled partial GRF.

Root Thread from MEDIA_OBJECT_WALKER

The root thread payload message for an MEDIA_OBJECT_WALKER command, which must be in Generic mode, has the same format as that of the generic mode root thread format.

Root thread payload layout for a MEDIA_OBJECT_WALKER command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	n/a	**Constants from CURBE when CURBE is enabled** m is a non-negative value
R(m+1)	R1	**In-line Data block**

The R0 header field is identical to that of Generic Mode Root Thread.

The inline data block field is the same as in the MEDIA_OBJECT command with zero-filled partial GRF.

There is no indirect data block field.

MEDIA_OBJECT_GRPID and MEDIA_OBJECT_WALKER with Groups Payload

The RO header of the MEDIA_OBJECT_GRPID & MEDIA_OBJECT_WALKER Payload with groups enabled:

DWord	Bits	Description
R0.7	31:0	**Group ID LSB.** This is the LSBs of the Group ID. For MEDIA_OBJECT_GRPID threads this is the entire group id. For MEDIA_OBJECT_WALKER threads the interpretation depends on the Group ID Loop Select: 0: No groups, field is 0. 1: cat(InnerGlobalCnt[6:0], OuterLocalCnt[9:0], MidLocalCnt[4:0], InnerLocalCnt[9:0]); rest of group

DWord	Bits	Description
		id is in R0.2. 2: cat(OuterGlobalCnt[6:0], InnerGlobalCnt[9:0], OuterLocalCnt[9:0], MidLocalCnt[4:0]); rest of group id is in R0.2. 3: cat(2'b0, OuterGlobalCnt[9:0], InnerGlobalCnt[9:0], OuterLocalCnt[9:0]). 4: cat(12'b0, OuterGlobalCnt[9:0], InnerGlobalCnt[9:0]). 5: cat(22'b0, OuterGlobalCnt[9:0]).
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power to which two is raised to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2		
	31	**Barrier ID MSB.** This field is bit[4] of the BarrierID; for full 5-bit barrier ID it should be combined with Barrier ID[3:0]. Format: U1
	30	**Media Group ID.** This dispatch originated in a Media_Walker or Media_Object_GrpId, and so the Thread Group ID is formed from either the loop counters (for Media_Walker) or a supplied group ID (for Media_Object_Grpid).
	29	**Barrier Enable.** This field indicates that a barrier has been allocated for this kernel.
	28	**SLM Enable.** This field indicates that Shared Local Memory has been allocated for this kernel.
	27:24	**BarrierID.** This field indicates the barrier that this kernel is associated with. There is a fifth bit for the BarrierID, in bit 31. Format: U4
	23:11	**Group ID MSB.** This field is the MSBs of the Group ID. These bits are 0 for MEDIA_OBJECT_GRPID. For MEDIA_OBJECT_WALKER threads the interpretation depends on the Group ID Loop Select: 0: No groups, field is 0. 1: cat(OuterGlobalCnt[9:0], InnerGlobalCnt[9:7]). 2: cat(10'b0, OuterGlobalCnt[9:7]). 3-5: All zero.
	10	Reserved: MBZ
	9:4	**Interface Descriptor Offset.** This field specifies the offset from the interface descriptor base pointer to the interface descriptor that is applied to this object. It is specified in units of interface

DWord	Bits	Description
		descriptors. Format = U6
	3:0	**Scoreboard Color.** This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
R0.1	31:27	Reserved: MBZ
	26:16	**Scoreboard Y.** This field provides the Y term of the scoreboard value of the current thread. Format = U11
	15:11	Reserved: MBZ
	10:0	**Scoreboard X.** This field provides the X term of the scoreboard value of the current thread. Format = U11
R0.0	31:30	Reserved: MBZ
	29:24	**SLM ID.** This field indicates the index to the SLM starting offset associated with this kernel. Format: U6
	23:16	**Scoreboard Mask.** Each bit indicates the corresponding dependency that the scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. **Bit n (for n = 0..7)**: Scoreboard n is dependent, where bit 23 maps to n = 0. Format = TRUE/FALSE
	15:0	**URB Handle.** This is the URB handle where indicating the URB space for use by the root thread and its children.

Thread Spawn Message

The thread spawn message is issued to the TS unit by a thread running on an EU. This message contains only one 8-DWord register. The thread spawn message may be used to:

- Spawn a child thread.
- Spawn a root thread (start dispatching a synchronized root thread).
- Dereference an URB handle.
- Indicate a thread termination, dereference other TS managed resource and may or may not dereference URB handle.
- Release a PRT_Fence.

To end a root thread, the end of thread message must be targeted at the thread spawner. In this case, the root thread sends a message with a "dereference resource" in the Opcode field. The thread spawner does *not* snoop the messages sideband to determine when a root thread has ended. Thread Spawner does not track when a child thread terminates, to be consistent a child thread should also terminate with a "dereference resource" message to the Thread Spawner. Software must set the Requester Type (root or child thread) field correctly.

TS dispatches one synchronized root thread upon receiving a 'spawn root thread' message (from a synchronization thread). The synchronizing thread must send the number of 'spawn root thread' message exactly the same as the subsequent 'synchronized root thread'. No more, no less. Otherwise, hardware behavior is undefined.

URB Handle Offset field in this message (in M0.4) has 10 bits, allowing addressing of a large URB space. However, when a parent thread writes into the URB, it subjects to the maximum URB offset limitation of the URB write message, which is only 6 bits (see Unified Return Buffer Chapter for details). In this case, the parent thread may have to modify the URB Return Handle 0 field of the URB write message to subdivide the large URB space that the thread manages.

Only a persistent root thread can use this message to dispatch a root thread if preemption exceptions are possible. The root thread requested by this message is not guaranteed to dispatch, and the persistent root thread must handle the case where it does not dispatch. When a context switch interrupt is recognized by the persistent root thread, all other root threads that had been dispatched have completed and no more will be dispatched.

Child threads requested by this message are guaranteed to dispatch in all cases, so long as the persistent root thread does not also dispatch synchronized root threads. A child thread does not dispatch if it is behind a synchronized root thread that is not dispatched due to a preemption exception.

In addition to monitor 'end of thread message' targeted to Thread Spawner, Thread Spawner also monitors the message targeting to Message Gateway for EOT signal. Therefore, a child thread, who doesn't hold any hardware resource (URB handle or scratch memory) that Thread Spawner manages, can terminate with a Gateway message with EOT on. The reason of this new TS feature is to avoid a possible risk condition as described below.

In a system running child threads, a parent thread is monitoring the status of the child threads by communications through Message Gateway. When a child thread is about to terminate, it sends a message to the parent through Message Gateway and then sends a second message of EOT (end of thread) to TS.

There is a latency between sending a message to parent thread and the EOT to TS due to message bus arbitration. The parent thread may acknowledge the GW message and issue a new child dispatch before the EOT was processed; basically threads are issued faster than retired.

Because the messages for new child dispatch and EOT go to the same queue in TS, if the queue gets full, EOTs will get blocked. In the case when all the EUs/Threads are full, this will create a system deadlock: no EOTs can be acknowledged by TS (to free up EU resource) and no child threads can be dispatched (to free up TS queue to receive EOT message).

Message Descriptor

The following table shows the lower 20 bits of the message descriptor within the SEND instruction for a thread spawn message.

Thread Spawn Message Descriptor

Message Payload

DWord	Bits	Description
M0.5	31:10	Ignored.
	9:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by a root thread upon thread completion. This field is valid only if the **Opcode** is “dereference resource”, and is ignored by hardware otherwise.
M0.4	31:16	Ignored.
	15:10	**Dispatch URB Length.** Indicates the number of 8-DWord URB entries contained in the Dispatch URB Handle that will be dispatched. When spawning a child thread, the URB handle contains most of the child thread's payload including the R0 header. When spawning a root thread, the URB handle contains the message passed from the requesting thread to the spawned “peer” root thread. The number of GRF registers that are initialized at the start of the spawned child thread is the sum of this field and the number of URB constants if present. The number of GRF registers that are initialized at the start of a spawned root thread is the sum of this field, the number of URB constants if present, and the URB handle received from VFE. This field is ignored if the Opcode is “dereference resource”. A Length of 0 can be used while spawning child threads to indicate that there is no payload beyond the required R0 header. A Length of 0 while spawning a root thread indicates that there is no payload at all from the parent thread. A spawned root has R0 supplied by the Media_Object command indirect/inline data. Format = U6 Range = [0,63] for child threads.
	9:0	**URB Handle Offset.** Specifies the 8-DWord URB entry offset into the URB handle that determines where the associated dispatch payload will be retrieved from when the spawned child or root thread is dispatched. This field is ignored if the **Opcode** is “dereference resource”. Format = U10 Range = [0,1023]
M0.3	31:0	Ignored.
M0.2	31	**Barrier ID MSB.** This field is bit[4] of the BarrierID; for full 5-bit barrier ID it should be combined with Barrier ID[3:0]. Format: U1
	30	Ignored.
	29	**Barrier Enable.** This field indicates that a barrier has been allocated for this kernel.
	28	**SLM Enable.** This field indicates that Shared Local Memory has been allocated for this kernel.
	27:24	**BarrierID.** This field indicates which one of the 16 Barriers this kernel is associated with. Format: U4
	23:16	Ignored.
	15:10	Ignored.

DWord	Bits	Description
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX). This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
M0.1	31:0	Ignored.
M0.0	31:24	**SLM ID.** This field encodes the SLM bank ID and SLM offset in the bank, for the shared local memory for the thread group.
	23:16	Reserved: MBZ
	15:0	**Dispatch URB Handle.** If **Opcode** (and **Requester Type**) is "spawn a child thread": Specifies the URB handle for the child thread. If **Opcode** (and **Requester Type**) is "spawn a root thread": Specifies the URB handle containing message (e.g. requester's gateway information) from the requesting thread to the spawned root thread. If **Opcode** is "dereference resource": This field is required on end of thread messages if the **Children Present** bit is set, as the handle must be dereferenced, otherwise this field is ignored.

Media-GPGPU Thread EOT Message

The thread EOT message is issued to the Gateway unit by a GPGPU or Media thread running on an EU. This message contains only one 8-DWord register. It indicates that the thread is terminating and the thread spawner should dereference the barrier and SLM resources that were optionally allocated with the thread when it was dispatched.

Message Descriptor

[Structure] Thread EOT Message Descriptor

Message Payload

DWord	Bits	Description
M0.5	31:8	Ignored.
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by a root thread upon thread completion.
M0.4	31:0	Ignored.
M0.3	31:0	Ignored.
M0.2	31	**Barrier ID MSB.** This field is bit[4] of the BarrierID; for full 5-bit barrier ID it should be combined with Barrier ID[3:0]. Format: U1
	30	Reserved: MBZ
	29	**Barrier Enable.** This field indicates that a barrier was allocated for this kernel.

DWord	Bits	Description
	28	**SLM Enable.** This field indicates that Shared Local Memory was allocated for this kernel.
	27:24	**BarrierID.** This field indicates which one of the 16 Barriers this kernel is associated with. There is a fifth bit for the BarrierID, in bit 31. Format: U4
	23:0	Ignored.
M0.1	31:0	Ignored.
M0.0	31:24	**SLM ID.** This field encodes the SLM bank ID and SLM offset in the bank, for the shared local memory for the thread group.
	23:0	Ignored.

L3 Cache and URB

This section describes the GFX L3 Cache, which is a large storage (introduced in IVB) that backs up various L2/L1 caches on many clients. It provides a simple way based partitioning option for each of a cluster of clients to get a dedicated chunk of the cache. It also acts as a GFX URB and can be configured as highly banked memory for EUs/ROWs.

L3 Cache and URB

This is the BSpec volume L3/URB/SLM.

LBCF Registers

Register
L3SQCREG1 - L3 SQCRegisters1
L3SQCregisters2
L3SQCregisters3
L3ControlRegister1
L3SLMRegister
L3SQCregister4
SCRATCH1
LTCDconfigregister
LBSconfigbits
L3BankStatus
LBCFconfigsavemsg

LNCF Registers

Register
ArbiterControlRegister
SCRATCHforLNCFunit
Eventselectionandbasecounters
EventSelectionandBaseCounters1
LPFCcontrolregister
LNCFconfigsavemsg
LNCFMOCSRegister0
LNCFMOCSRegister1
LNCFMOCSRegister2
LNCFMOCSRegister3
LNCFMOCSRegister4
LNCFMOCSRegister5
LNCFMOCSRegister6
LNCFMOCSRegister7
LNCFMOCSRegister8
LNCFMOCSRegister9
LNCFMOCSRegister10
LNCFMOCSRegister11
LNCFMOCSRegister12
LNCFMOCSRegister13
LNCFMOCSRegister14
LNCFMOCSRegister15
LNCFMOCSRegister16
LNCFMOCSRegister17
LNCFMOCSRegister18
LNCFMOCSRegister19
LNCFMOCSRegister20
LNCFMOCSRegister21
LNCFMOCSRegister22
LNCFMOCSRegister23
LNCFMOCSRegister24
LNCFMOCSRegister25
LNCFMOCSRegister26
LNCFMOCSRegister27
LNCFMOCSRegister28
LNCFMOCSRegister29
LNCFMOCSRegister30
LNCFMOCSRegister31

LPFC Registers

Register
PMflushandslmsaverestore
GlobalInvalidationRegister
PowercontextSaveRegisterforLPFC
FirstBufferSizeandStart
SecondBufferSize
ErrorReportingRegister
FramecountandDrawcallnumber
SaveTimer
Masterstarttimer

Overview

The changes to L3 for SKL are minor. The overall organization of the L3 and theory of operation have not changed. It retains all the performance enhancements from previous generations with some additional improvements.

To provide the bandwidth needed L3 is still organized into independent banks which can be accessed concurrently. Clocking remains 2X in the arrays to balance bandwidth with incoming requests. The L3$ and URB data for all L3 banks are shared as a contiguous memory space for all products regardless of how many banks or slices.

- Each Bank 192KB in size.
- Each logical bank consists of:
    - Data Array
    - Tag Array
    - LRU Array (implements a Pseudo Least Recently Used algorithm)
    - State Array
    - SuperQ Buffer
    - Atomic Processing Units
- The rest of the support logic around L3 consists of:
    - SuperQ (main scheduler).
    - Ingress/Egress queues to L3/SQ (L3 arbiter).
    - CAM structures to maintain coherency.
    - Crossbars for data routing.
- Use of 2x/1x clocking.
- L3 can operate concurrently in GFX and IA coherent domain.
- A portion of L3 can be allocated as highly banked memory and/or unified buffer (URB).

L3 Bank Configuration

Each L3 bank is identical as described below. In products where there is only a single L3 bank supported, the bank will be different to support a larger data array. The Multi-bank describes the L3 Bank configuration for all other products.

Multi-Bank: 192KB data-array and 96 logical ways:

- Up to 64 ways, up to 128KB, tagged for L3$, remaining is treated as memory.
- 64KB or 96KB allocated for URB (does not use tag).
- Shared local memory (SLM) capable (64KB)
    - Uses Highly-banked memory to allow up to 16 32-bit accesses in parallel within a 64KB space per sub-slice.
    - Highly-banked memory formed from 16x4KB arrays

All Bank Implementations:

- 64B Cacheline.
- Interface 64B to SQDB for the fill/write path, 64B Read/Evict path to SQDB. Additional 64B read and 64B write capability for SLM.
- Data protection via ECC.
- TAG/LRU/STATE (using gen-ram via RLS flows):
    - 39/48-bit addressing support in TAG.
    - 6-bit state (2-bits of MESI, 2-bits of Surface type, 1 bit Phys/Virtual, 1-bit Global/Local).
    - Intel pseudo-LRU implementation for selecting the line to be replaced or 1b LRU (added for Gen9).

Bandwidth and Throughput Capability

Programming Note	
Context:	Bandwidth and Throughput Capability

Note: The table below shows the throughput capability for the L3/URB and SLM per bank. Note that L3$ and URB share the same pipelines, so the throughput for L3$ and URB is shared as well. SLM has dedicated pipelining and therefore has its own throughput capability. Total system throughput is derived by multiplying the throughput numbers below by the total number of L3 Banks supported by a product. There is an additional limitation on throughput based the source of the transactions. A single client (e.g. an HDC or sub-slice) can only issue one 64-byte read or write per clock.

Memory Type	Read Throughput (Bytes/Clock)	Write Throughput (Bytes/Clock)	Atomic Throughput (32b Ops/Clock)
L3$ or URB	64	64	16
SLM	64	64	16

L3 Blocks Overview

L3 is formed via some number of logical banks that are identical to each other. The major blocks in each logical bank are:

- L3 Cache Arrays & Controller
- Super Q and related data buffer
- Ingress queues and related CAMs with arbitration
- Atomics Block/SLM pipeline & crossbar for data routing

Our vision is to build a compute scalable cache where with each additional compute both the size and bandwidth are scaled while maintaining the functional single cache concept. Each added bank becomes an additional cache rather than an independent content. The concept is to be able to keep a single copy of a line and service all requesters via distributing their accesses over many physical caches.

Size of L3 Bank and Allocations

Multi-Bank Allocation Options

The table below shows the size of the L3 bank and the ranges of allocation for L3$, URB, and SLM. Below is a table of all validated configurations supported.

No Shared Local Memory Mode (KBytes)						
			L3			
Config	SLM	URB	Rest	DC	RO(I/S/C/T)	Sum
0	0	96	96	0	0	192
1	0	96	0	32	64	192
2	0	64	0	128	0	192
3	0	64	0	0	128	192
4	0	64	128	0	0	192
Shared Local Memory Mode (KBytes)						
			L3			
Config	SLM	URB	Rest	DC	RO(I/S/C/T)	Sum
1	64	32	96	0	0	192
2	64	32	0	32	64	192
3	64	32	0	64	32	192
4	64	16	112	0	0	192

Essentially, the L3 block can either support SLM or not. When SLM is supported, each slice provides 64KB of its 192KB data array for SLM. When SLM is not supported, the 64KB is allocated to URB or split equally between URB and L3$.

The number of L3 Banks will vary for different products and SKUs. The number of banks supported for each product is defined in the Configurations section of the BSPEC. The total amount of L3$, URB, and SLM supported by a product can be calculated by multiplying the number of banks by the values in the above tables.

L3 Cache Theory of Operation

L3/URB operation is required for various clients to access L3 as their back-up cache or memory space. The following clients are listed as L3/URB clients:

L3 Clients:

- Data Cluster (i.e. spill/fills, load/stores, global memory accesses, ...) (read/write)
- Sampler (L2$ - MT) (read)
- IME (motion estimation) (read)
- I$ (Instruction Cache) (read)
- State Arbiter (L3 is state cache replacement) (read)
- Constant cache (read)
- Tesselator (always un-cacheable for SW tessellation)

URB Clients:

- TD-L (read client) – Local Thread Dispatcher
- SBE (read client) – SF Backend
- SOL (read client) – Stream Out
- CL (read client) – Clipper
- GS (read client) – Geometry Shader
- TE (read Client) – Tessalator
- DC (read/write client) – Data Cluster
- VF/(VFE/CS) (write client) – VF acts on behalf of all

L3 vs. URB accesses are separated with a simple field on the request field, for most clients this is only one direction with the exception of data port where both could be addressed L3 or URB. The destination field is part of the request and could be set to re-direct the request to:

- L3 cache
- URB
- State Arbiter
- SLM (highly banked memory)

L3 access/cacheability is determined via request field as well, such parameter will be part of the surface state or base address programming of L3 clients and will be communicated to L3 Cluster along with the request packet.

SLM is accessed in parallel with L3$/URB accesses. L3$/URB share the normal array, and must be serialized. SLM accesses have a dedicate highly-banked array which has lower latency and is not serialized with L3$/URB accesses.

L3 Cache Theory of Operation

L3/URB operation is required for various clients to access L3 as their back-up cache or memory space. The following clients are listed as L3/URB clients:

L3 Clients
Data Cluster (i.e. spill/fills, load/stores, global memory accesses, ...(read/write)
Sampler (L2$ - MT) (read)
IME (motion estimation) (read)
I$ (Instruction Cache) (read)
State Arbiter (L3 is state cache replacement) (read)
Constant cache (read)
Tesselator (always un-cacheable for SW tessellation)

URB Clients:

- TD-L (read client) – Local Thread Dispatcher
- SBE (read client) – SF Backend
- SOL (read client) – Stream Out
- CL (read client) – Clipper
- GS (read client) – Geometry Shader
- TE (read Client) – Tessalator
- DC (read/write client) – Data Cluster
- VF/(VFE/CS) (write client) – VF acts on behalf of all

L3 vs. URB accesses are separated with a simple field on the request field, for most clients this is only one direction with the exception of data port where both could be addressed L3 or URB. The destination field is part of the request and could be set to re-direct the request to:

- L3 cache
- URB
- State Arbiter
- SLM (highly banked memory)

L3 access/cacheability is determined via request field as well, such parameter will be part of the surface state or base address programming of L3 clients and will be communicated to L3 Cluster along with the request packet.

SLM is accessed in parallel with L3$/URB accesses. L3$/URB share the normal array, and must be serialized. SLM accesses have a dedicate highly-banked array which has lower latency and is not serialized with L3$/URB accesses.

Access Policy

L3 partitioning has been reduced to remove some of the complexity. The read-only space clients are merged to have combined space which is allocated over the entire L3 cache and shared with read/write space. The Global vs Local distinction is removed with the coherency being updated.

The destination field should reflect:

- L3 R/W data
- L3 RO data
- URB
- SLM
- State ARB (SARB)

Arbitration Policies

There are three types of L3 transactions which are arbitrated between for Tag access: Snoops, Fills, and Reads (writes are done as full cache-line writes and therefore are treated like fills). Snoops requests arrive from a dedicated queue (LBS), and Fill requests arrive from the LSQC and both are deposited in dedicated ingress FIFOs. Read requests arrive from the LSQC but are sent to two parallel ingress FIFOs. So, at any given time one snoop, one fill, and two reads can be requesting tag access. Once a request is submitted from the LSQC or LBS to the Tag Ingress FIFOs, they cannot be re-ordered. Only the head of the FIFOs is arbitrated against. This is sub-optimal, but much simpler than attempting to re-order within the ingress FIFOs.

The transactions are arbitrated in the 1x clk domain but are submitted into the 2x clk domain (the tag, state, LRU, and data arrays operate in the 2x clk domain). So, up to two commands can be submitted to the tag pipeline in a single 1x clk.

Tag Request Arbitration

The arbitration for tag access is based on cycle type and set address:

- Snoops have highest priority. Whenever a snoop is presented, it takes precedence over Fills and Reads. Only one snoop can be done at a time (per 1x clk).
- Fills have priority over Reads. Only one fill can be done at a time.
- Reads/Atomics take any empty arbitration slots. Since two commands can be submitted for each 1x clock, that means up to two reads can be submitted at a time (per 1x clk).
- No two transactions to the same set address are allowed. In this case one of the ports will be held idle to insert a bubble in the pipeline. This set address dependency is caused by resource conflicts

in both the Tag and LRU arrays. For Gen9, special buffers will be added to contain results from the last one or two aromic operations, to allow for back-to-back atomics to the same CL. This eliminates the need for data array accesses, which eliminates a bubble in the data pipelines.

- Only L3 accesses request Tag access. URB and SLM requests go directly to the data array pipeline.

Data Request Arbitration

The data-array is accessed by the LSQD whenever a read operation (for data read or eviction) or fill is required. The LSQD receives commands from the LSQC based on the tag response (in the case of L3 accesses) or for URB accesses without any need for tag lookup.

SLM data accesses come from a separate pipeline. However, SLM accesses are always to the dedicated highly-banked memory and do NOT require arbitration against L3/URB requests. In the case where SLM is not enabled, then the highly-banked memory is allocated to URB. In this case, data returns from the highly-banked memory will need to be muxed with the data from the larger data array.

L3 (from the Tag pipeline), URB, and Snoop-Evictions must be arbitrated for access to the data array. The arbitration is a fixed priority:

with the Snoops having highest priority and URB having next highest priority.

Back-to-back reads and back-to-back writes to the data array will always be prevented through arbitration. A 2-clock (2x clk) bubble will be inserted to allow the data array to recover and precharge as required.

The highly banked memory still has the same back-to-back read or back-to-back write restriction.

L3 Ordering Restrictions

The LSQC will enfore specific ordering requirements on the accesses to the L3/URB, but will still allow out-of-order accesses where possible.

Basic Ordering Requirements

Requirements:

1. Ordering of transactions primary purpose is to ensure coherency and causality for software accesses to memory. For example, if a thread writes to a memory address, it expects that any subsequent read issued by the same thread to the same address should read back what it wrote. Conversely, if the read is before the write, it should not read back what it wrote.
2. Transactions which have hardware constraints must be ordered to avoid conflicts. For example, two FP-Atomic Add operations must be ordered to avoid conflict on a single FP Adder.

With these two requirements in mind, the order requirements are defined.

Rules:

1. All reads/fills and atomics to the same cacheline must be ordered and serialized such the oldest is completed first and the youngest is completed last. This means that an older read/fill in the SQ must be completed prior to a younger read/fill being started. This meets requirement #1 above. Strictly speaking two or more reads to the same cacheline need not be ordered, but there's no particular advantage to re-ordering these.
2. All reads/fills/atomics to a cacheline can proceed even if an eviction is pending for that cacheline. In practice, holding a read/fill until an eviction is completed is not necessary because the L3 allocation policy (see next section) is such that allocation of a new line into a set is not done until the Fill arrives. So, any read/fill to the old (to be evicted) line can proceed normally. Once the Fill does arrive, the eviction of the line and update of the tag will make any subsequent read/fill a miss and cause that transaction to reallocate through the normal flows.
3. Two or more evictions of the same cacheline must be ordered with the oldest going first and the youngest going last. Rule #2 above creates a funny boundary case where you can have two evictions to the same cacheline in the LQSC at the same time. This is possible in a scenario where you have a dirty fill which causes an eviction to a cacheline in a set, followed by a long series of dirty evictions to the same set. Eventually, if the Fill for your first line is taking a very long time to return, it is possible for a 2nd eviction to that same line to occur. For coherent lines, this cannot occur because any coherent cycle would issue a self-snoop which would cause the eviction to be pushed prior to any subsequent eviction (see Rule #5).
4. All FP-atomics to any cacheline are ordered and serialized. This is done to break the conflict on use of the Floating-Point ALU hardware. This meets requirement #2 above.
5. All snoops must be ordered with respect to evictions to the same line. Snoops have their own queue, but they also check the contents of the SQ for evictions to the same line. In other words, snoops should push older evictions to the same cacheline.

This is summarized below. Empty cells denote "no ordering/serialization required". A "Yes" indicates an ordering/serialization between the transactions on the row and column. The "Same CL" indicates that the ordering/serialization is only for transactions to the same cache-line.

Transaction Ordering Rules for Transactions to the L3/URB

Transaction	Read/Fill	Eviction	Int. Atomic	Snoop	FP-Atomic
Read/Fill	Yes, same CL		Yes, same CL		Yes, same CL
Eviction		Yes, same CL			
Int-Atomic	Yes, same CL		Yes, same CL		Yes, same CL
Snoop		Yes, same CL			
FP-Atomic	Yes, same CL		Yes, Same CL		Yes

Note, that the ordering/serialization described above does not limit accesses to the TAG. When two accesses arrive to the same cache-line in the SuperQ, we still form the parent-child link to allow for serialization of the older before the younger, however the younger can still proceed with TAG look-up as soon as older sends its TAG look up to L3 without waiting for the result of the older. If older is a HIT, the child will get a HIT as well hence no issue. If older is a MISS, the younger will get a MISS as well. In this case the older is the only one launching a read (FILL) to memory. Younger will be stalled until older completes the FILL. Child will replay the TAG look up after FILL operation. If there are more CHILD accesses they will immediately play once the second CHILD is sent to TAG pipeline. This optimization allows for a reduction in overall latency for pending requests by allowing tag look-ups for HITs to be done ahead of time.

Allocation Policy

During the MISS time of L3 look-up there is no pre-allocation. Only after the miss data is returned from LLC/Memory, the fill will be sent to data array (it is guaranteed not a match due to single address processing) and allocation will take place during the Fill time.

If the allocated entry is not modified, than FILL will overwrite the location and pipeline moves on. If the allocated entry carries a dirty data, eviction takes place and dirty data will be moved to SQ for writing out to memory. The Tag and state will only be updated once the fill and any eviction is actually being done.

The read for eviction and the fill operation are considered atomic. Only one transaction is issued from the super Q, but it creates a Read followed by a Fill. Victim data is returned to the SQDB.

Allocation on Fill eliminates many boundary cases via regulating the entry invalidation at the last phase of the data servicing. However, it also creates boundary cases which are handled in the LSQC.

High Level Algorithm Descriptions

The pLRU algorithm uses a binary decision tree with (N-1) nodes. Each node is controlled by a single bit. When a way is filled, all the nodes in the tree to reach that way are flipped to point to another way which is essentially as far away as you can get. The new cache-line is guaranteed to be retained for N-2 additional fills to the set. The LRU bits are only updated on Fills.

An N-way 1b LRU has an N-bit vector for each set. The vector is initialized to all 0's. As each Fill request arrives, the first bit in the vector with a 0 is selected, that way is replaced, and the bit is flipped. Each subsequent fill will search for the first 0 and replace that line and flip it's bit. Eventually, when all N bits are 1 and there are no candidates, all bits are cleared and the first way is selected.

Both implementations use approximately the same amount of logic with the N-1 bits per bits. One uses a binary tree to decode and selects bits, whereas the other has to handle read-hits.

High Level Algorithm Descriptions

The pLRU algorithm uses a binary decision tree with (N-1) nodes. Each node is controlled by a single bit. When a way is filled, all the nodes in the tree to reach that way are flipped to point to another way which

is essentially as far away as you can get. The new cache-line is guaranteed to be retained for N-2 additional fills to the set. The LRU bits are only updated on Fills.

An N-way 1b LRU has an N-bit vector for each set. The vector is initialized to all 0's. As each Fill request arrives, the first bit in the vector with a 0 is selected, that way is replaced, and the bit is flipped. Each subsequent fill will search for the first 0 and replace that line and flip it's bit. Eventually, when all N bits are 1 and there are no candidates, all bits are cleared and the first way is selected.

Both implementations use approximately the same amount of logic with the N-1 bits per bits. One uses a binary tree to decode and selects bits, whereas the other has to handle read-hits.

Memory Object Control State on Cacheability

Memory Object Control State information has been consolidated under the memory view section of the BSpec.

Overview

The changes to L3 for SKL are minor. The overall organization of the L3 and theory of operation have not changed. It retains all the performance enhancements from previous generations with some additional improvements.

To provide the bandwidth needed L3 is still organized into independent banks which can be accessed concurrently. Clocking remains 2X in the arrays to balance bandwidth with incoming requests. The L3$ and URB data for all L3 banks are shared as a contiguous memory space for all products regardless of how many banks or slices.

- Each Bank 192KB in size.
- Each logical bank consists of:
    - Data Array
    - Tag Array
    - LRU Array (implements a Pseudo Least Recently Used algorithm)
    - State Array
    - SuperQ Buffer
    - Atomic Processing Units
- The rest of the support logic around L3 consists of:
    - SuperQ (main scheduler).
    - Ingress/Egress queues to L3/SQ (L3 arbiter).
    - CAM structures to maintain coherency.
    - Crossbars for data routing.
- Use of 2x/1x clocking.
- L3 can operate concurrently in GFX and IA coherent domain.
- A portion of L3 can be allocated as highly banked memory and/or unified buffer (URB).

Atomics

An atomic operation may involve both reading from and then writing to a memory location. Atomic operations apply only to either u# (Unordered Access Views) or g# (Thread Group Shared Memory). It is guaranteed that when a thread issues an atomic operation on a memory address, no write to the same address from outside the current atomic operation by any thread can occur between the atomic read and write.

If multiple atomic operations from different threads target the same address, the operations are serialized in an undefined order. This serialization occurs due to L3 serialization rules to the same address.

Atomic operations do not imply a memory or thread fence. If the program author/compiler does not make appropriate use of fences, it is not guaranteed that all threads see the result of any given memory operation at the same time, or in any particular order with respect to updates to other memory addresses. However atomic operations are always stated on a global level (except on shared local memory), when atomic is operation is complete final result is always visible to all thread groups.

Each generation since Gen7 has increased the capability and performance of atomic operations.

Gen8 atomics introduced these features:

- Double size operands where 8B atomic operations are introduced for 64-bit data types. There is also an addition 16B "atomic_cmp/wr16B" to the table. All these new operands apply to global memory only.
- Floats for 4B/8B accesses, only floating point adder is used.
    - (*note: if L3 atomics are disabled, floating point atomics can be used*)
- Move global atomic ops to L3 (keep the GTI support for non-coherent L3 mode) and share same atomic OPs as SLM.
    - GT3 can process up to 192 IA coherent atomics per cycle.

Gen 9 adds these features:

- Hardware improves the performance of atomics to the same cacheline. Specifically, back-to-back atomics to the same memory location are done without the need to re-fetch the result of the previous atomic from L3. This should increase bandwidth for atomics to the same location significantly over Gen8/BDW.
- Single-Precision FP Min/Max and Compare/Exchange Instructions are added.

Gen 9 atomics hitting an overflow condition for memory buffer may corrupt other atomics.

Programming Note	
Context:	Atomics
API Specification: The API specification says that atomic operations on Thread Group Shared Memory are atomic with respect to other atomic operations, as well as operations that only perform reads (loads). However atomic operations on Thread Group Shared Memory are *not*atomic with respect to operations that perform only writes	

Programming Note	
Context:	Atomics
(stores) to memory. Mixing atomics and stores on the same Thread Group Shared Memory address without thread synchronization and memory fencing between them produces undefined results at the address involved. This restriction arises because some implementations of loads and stores do not honor the locking semantics for implementing atomics. It turns out this has no impact on loads, since they are guaranteed to retrieve a value either before or after an atomic (they will not retrieve partially updated values, given they are all defined at 32-bit quanta). However store operations could find their way into the middle of an atomic operation and thus have their effect possibly lost.	

For Gen9 we do NOT allow stores to break an atomic.

In L3 or SLM, the atomic operation leads to a read-modify-write operation on the destination location with the option of returning value back to requester. The table below is defined as a list of atomic operations needed:

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
Atomic_AND	0000_0001	Single component 32-bit bitwise AND of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	“old_dst” AND “src0”	global/SLM	old_dst
Atomic_OR	0000_0010	Single component 32-bit bitwise OR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	“old_dst” OR “src0”	global/SLM	old_dst
Atomic_XOR	0000_0011	Single component 32-bit bitwise XOR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	“old_dst” XOR “src0”	global/SLM	old_dst
Atomic_MOVE	0000_0100	Replace the dst with src0.	“src0”	global/SLM	old_dst
Atomic_INC	0000_0101	Single component 32-bit integer increment of dst back into dst.	“old_dst + 1”	global/SLM	old_dst
Atomic_DEC	0000_0110	Single component 32-bit integer decrement of dst back into dst.	“old_dst - 1”	global/SLM	old_dst
Atomic_ADD	0000_0111	Single component 32-bit integer add of operand src0 into dst at 32-bit per component address dstAddress, performed atomically. Insensitive to sign.	“old_dst + src0”	global/SLM	old_dst
Atomic_SUB	0000_1000	Single component 32-bit integer subtraction of operand src0 from dst at 32-bit per component address	“old_dst - src0”	global/SLM	old_dst

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
		dstAddress, performed atomically. Insensitive to sign.			
Atomic_RSUB	0000_1001	Single component 32-bit integer subtraction of operand dst from src0 into dst at 32-bit per component address dstAddress, performed atomically. Insensitive to sign.	“src0 - old_dst”	global/SLM	old_dst
Atomic_IMAX	0000_1010	Single component 32-bit signed MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMAX(old_dst, src0)	global/SLM	old_dst
Atomic_IMIN	0000_1011	Single component 32-bit signed MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMIN(old_dst, src0)	global/SLM	old_dst
Atomic_UMAX	0000_1100	Single component 32-bit unsigned MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMAX(old_dst, src0)	global/SLM	old_dst
Atomic_UMIN	0000_1101	Single component 32-bit unsigned MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMIN(old_dst, src0)	global/SLM	old_dst
Atomic_ CMP/WR	0000_1110	Single component 32-bit value compare of operand src0 with dst at 32-bit per component address dstAddress.	(src0 == old_dst)?	global/SLM	old_dst
		If the compared values are identical, the single-component 32-bit value in src1 is written to destination memory, else the destination is not changed.	src1:		
		The entire compare+write operation is performed atomically.	old_dst		
Atomic_ PREDEC	0000_1111	Single component 32-bit integer decrement of dst back into dst.	“old_dst - 1”	global/SLM	new_dst
Atomic_AND8B	0010_0001	Single component 64-bit bitwise AND of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	“old_dst8B” AND “src08B”	global	old_dst8B

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
Atomic_OR8B	0010_0010	Single component 64-bit bitwise OR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	“old_dst8B” OR “src08B”	global	old_dst8B
Atomic_XOR8B	0010_0011	Single component 64-bit bitwise XOR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	“old_dst8B” XOR “src08B”	global	old_dst8B
Atomic_ MOVE8B	0010_0100	Replacement of the dst with src0.	“src08B”	global	old_dst8B
Atomic_INC8B	0010_0101	Single component 64-bit integer increment of dst back into dst	“old_dst8B + 1”	global	old_dst8B
Atomic_DEC8B	0010_0110	Single component 64-bit integer decrement of dst back into dst	“old_dst8B - 1”	global	old_dst8B
Atomic_ADD8B	0010_0111	Single component 64-bit integer add of operand src0 into dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	“old_dst8B + src08B”	global	old_dst8B
Atomic_SUB8B	0010_1000	Single component 64-bit integer subtraction of operand src0 from dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	“old_dst8B - src08B”	global	old_dst8B
Atomic_ RSUB8B	0010_1001	Single component 64-bit integer subtraction of operand dst from src0 into dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	“src08B - old_dst8B”	global	old_dst8B
Atomic_IMAX8B	0010_1010	Single component 64-bit signed MAX of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMAX (old_dst8B, src08B)	global	old_dst8B
Atomic_IMIN8B	0010_1011	Single component 64-bit signed MIN of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMIN (old_dst8B, src08B)	global	old_dst8B
Atomic_	0010_1100	Single component 64-bit unsigned MAX of operand src0 into dst at 64-bit per component address dstAddress,	UMAX (old_dst8B, src08B)	global	old_dst8B

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
UMAX8B		performed atomically.			
Atomic_ UMIN8B	0010_1101	Single component 64-bit unsigned MIN of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	UMIN (old_dst8B, src08B)	global	old_dst8B
Atomic_ CMP/WR8B	0010_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress.	(src08B == old_dst8B)?	global	old_dst8B
		If the compared values are identical, the single-component 64-bit value in src1 is written to destination memory, else the destination is not changed.	src18B:		
		The entire compare+write operation is performed atomically.	old_dst8B		
Atomic_ PREDEC8B	0010_1111	Single component 64-bit integer decrement of dst back into dst.	“old_dst8B - 1”	global	new_dst8B
Atomic_ CMP/WR16B	0100_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress.	(src0_16B == old_dst16B)?	global	old_dst16B

L3 Coherency

Coherency is one of the crucial topics within L3, there are multiple levels of coherency that are checked and ensured via L3. The list of domains and flows is dependent on the usage models however basic premise is always the same.

The coherency levels:

1. Thread Level Coherency within a Thread Group
2. Thread Group Coherency between multiple domains
3. GPU/IA level coherency

Besides these special domains basic producer/consumer models are followed which are listed as:

1. Fixed function is producing
2. Data Port is producing

All these high level coherency models are investigated and addressed.

Thread Level Coherency

A given thread group is contained within a quarter slice in Gen9, where its writes and reads target the L3 for global memory and SLM for shared local memory. Given the shared local memory view is the same for all quarter slice accesses there is no question about the coherency or data sharing within the thread group. Local syncs are executed up to HDC boundary and not exposed to L3.

Thread Group Coherency

Thread groups can be distributed to multiple quarter slices that are physically far from each other. The coherency between thread groups can only be maintained for their global memory accesses. There are two implications of this coherency depending on the mode we are operating at:

1. Non IA-Coherent L3 mode: This is the same methodology that was introduced on Gen7.5 with the addition of GT4 support. The cross thread group coherency is maintained via Sync Global which is processed by L3 as well as introducing WT mode to be able to update global memory with latest data. This mechanism is supported on Gen9, but not expected to be used according to feedback from software architects.
2. IA-Coherent L3 mode: For the new mode of operation, there is no need to have WT behavior. The data in L3 is already visible to all consumers (i.e. other thread groups of GPU or IA cores). Sync'global has no affect given the data in L3 is already globally visible to all consumers.

GPUIA Level Coherency

In non-coherent L3 mode (i.e. gen7.5 behavior), data sharing between GPU and IA happens via a s/w controlled flow which requires the internal GPU caches to be flushed in order to make the data visible, similarly same caches need to be invalidated when IA produces data. This mode of operation will still be available for gen9. Within the L3, non-coherent accesses use "virtual" addresses and are tagged in the L3 Tag array using the Virtual/Physical bit.

In Gen9, coherent memory is supported where L3 content is visible to IA via snoops. This allows certain streams (i.e. data port) to be GO (Globally Observable) for IA once posted to L3, allowing shorter loop for completions and eliminating the need to do an entire pipeline flush over L3 to get it sync'ed to IA coherent domain. Coherency enhances the general programmability of GPU when cooperating with IA. Within the L3, coherent accesses use a full 48-bit physical address and are tagged using the Virtual/Physical bit in the I3 Tag array.

Coherency Usage Models

This section is to give some examples of usage models and high level handling within L3. They are specific to L3 flows and not meant to represent over coherency usage models on the system level.

Fixed Func Producing (URB)

Fixed functions clients and slice clients consuming is a very common usage model for URB based data sharing. In this mode, FF sends writes to URB and shifts to their pipeline, rest of the pipeline clients including slice clients will read their content from URB.

Process is achieved via URB writes completion from the GO ("Globally observable") point which would be the *slice node* for gen9. Once slice node consumes the write and schedules for BANK the completion is returned back to producer which enables the consumer. At the time the consumers start, the data is already at the point of GO where BANK superQ will keep the ordering.

Note that a node should not be scheduling writes from ingress queues if there are no credits and no progress can be made. The completion from a node is only given once the write from the producer leaves the node ingress queue.

Fixed Func Producing (Push Constants)

Push constants are also similarly processed; in fact the target is the same: URB. However URB accesses are processed with their URB offsets via TAG, their original address is based on virtual memory pointing to buffer location in memory. The delivery mechanism makes the CAMs not possible hence the completions are managed once access is deemed towards SuperQ of the corresponding L3 Bank.

There are additional constraints independent of L3 handling of push constants which are defined as part of the Global Arbitration fabric.

EUs Producing via HDC

Data port is the producer for Global and Shared Local memory types. The shared local memory is specific to a data port, hence L3 does not have to do anything special to maintain coherency but to keep accesses to SLM in-order.

For Global memory coherency is maintained via combination of many mechanisms.

1. Completion tracking: Each HDC tracks their writes towards L3 and wait for them to reach to GO. GO message is given by L3

a. For Non-coherent/virtual addressed Write: Once the ingress queue retires the write in the corresponding node to targeted bank.
b. For Coherent/physical addressed Write: Once the ownership is obtained for the write (i.e. read-for-ownership or invalid-to-modified) which means write is GO with respect to IA cores.

GO information is used within data port to make sure handle releases are gated until the producers updates are globally visible.

2. Thread Level Flush: EU threads have the capability to push globally tagged data from L3 to next level caches. The capability was added as part of Gen7.5 GT4 development for synchronizing cross contents between two cores. It is still supported in Gen9, but it is not used according to software architects. Usage is not recommended.

Invalidation and Flushes

For Gen9, there are two different sources to initiate flushes and invalidations:

- Command Streamer initiated
- EU/thread initiated

Both cases have two modes of operation

- Non-IA coherent L3
- IA coherent L3

In addition there are side flows that back up the various flows and they do require invalidation/flush sequences to be executed for their purpose. Their behavior has no impact between two modes of L3 operation.

- Global Invalidation
- Power Management Invalidation

Node Invalidation Architecture

The idea behind the node architecture is to make each L3 node to be independent and easily scalable; this is done via standardizing the flows and using common interfaces to exchange information between blocks. Invalidations and Flushes are meant to be tuned to similarly to fit into the concept. The final result of invalidation is that all cache-lines related to a surface or type will be invalid in the cache and all modified lines will have been written back to system memory.

Invalidation within a node may be done on a single surface or it may involve the entire cache depending on how the cache is configured and what type of invalidation is being requested. It is up to the node to figure this out and do the right thing. For example, read-only lines will never be written back since they cannot be modified, so they will only be invalidated. Modified read/write lines will be explicitly written back.

Each node performs the invalidation of all candidate lines for its own slice. This is coordinated by the unslice, and invalidation requests are submitted to each slice. Each slice indicates completion back to the unslice.

Each node will have to be notified on all invalidations as soon as possible and their behavior is to make that particular stream un-cacheable even if it did not start the invalidation process already. The process is much simpler when it comes to operations which do not require a feedback back to originator. The command streamer top of the pipe invalidations are under this category. Each node can process these invalidations independently but act on it for turning the particular stream to look uncacheable right away.

For the invalidation and flush processes where feedback is required, the event needs to be synchronized between nodes.

Each node receiving the invalidation or flush will push their ingress FIFOs to make sure their content is also covered as part of the flush event.

Command Streamer Invalidation Flows

Command streamer can do top of the pipe invalidations which are direct connections from command streamer unit to L3 in respective slices, as well as pipeline flush which runs through the data port to respective slice L3 node. Both flows are handled separately.

Any context can have both coherent and non-coherent L3 entries. Hence L3 has no register bit that says it is running in coherent vs non-coherent mode. In the operation time, both modes will co-exist simultaneously where some lines in the cache are following a coherent protocol (i.e. physical address) and remaining lines are following non-coherent protocol (i.e. virtual address).

Non-IA Coherent Flows

This is the traditional flow where the content of L3 needs to be invalidated or flushed similar to gen7.5 flows.

Top of the Pipe Invalidations

For this case, the invalidation causes the cache to drain all FIFOs and pipelines, and the node performs all invalidations. Only after completion in all slices will new transactions be sent to the L3.

The Top of the Pipe invalidation is intended for cases where invalidation and completion need to be coordinated between slices. Therefore, each slice performs invalidation, but will not proceed until all slices have completed the invalidation.

Nodes are responsible to serialize the invalidation and use double buffering for each event.

Pipeline Flush

Pipeline invalidation is much simpler and optimized for performance. In this case there is no need to coordinate between slices and the operation is more like a “sync” point. Transactions arriving after the invalidation request will still be queued in the SQ, but will only be processed after the invalidation of all candidate lines in the cache is complete. The invalidation acts like a fence.

Nodes are responsible to serialize the invalidation and use double buffering for each event.

IA-Coherent Flows

In principle IA-coherent flow is same from node perspective, the only difference is within the banks given the data that we are trying to flush or invalidate is already coherent with IA. Such case eliminates the need to push any bank content explicitly out to LLC.

All that is needed from the bank is to make sure LSQC content is posted into the bank where it is IA visible for a bank to return flush/invalidation completion status to the node.

EUThread Flows

As part of gen9, we continue to support the capability for EUs to be able to perform flush/invalidation events that are not coordinated between other EUs. HDC will relate the request to corresponding L3 and require the entire content of this buffer to be invalidated or flushed to relative coherency domain.

Similar to pipeline flush and top of the pipe invalidations, each node receiving this event will have to communicate with other nodes and make the corresponding traffic un-cacheable before returning the response back to corresponding HDC, allowing progress. The actual invalidation/flush will be deferred and performed once all nodes agree on what needs to be done.

The IA-coherent vs non-IA coherent treatment is same as command streamer flows and still applicable for EU/Thread Flows for invalidation/flush.

Global Invalidation

As part of a mitigation plan to plug holes, a global invalidation flow is introduced where each L3 will get the request from their SARB (or config agent) and again coordinate between the nodes. Once invalidation/flush is performed, completion will be returned back to config agent to clear the flags allowing s/w to observe the end of the global invalidation. Only lines which are tagged as “global” will be flushed in this manner.

Global invalidation will invalidate/flush every line which has its global bit set regardless of whether the line is tagged as virtual or physical (non-coherent or coherent).

Power Management Invalidation

Given we have a IA-coherent mode where a standard pipeline flush does not push the modified lines to outside of GT and we are progressing to deferred invalidations, we need a way to ensure L3 content is cleared and deferred events are complete. This is where PM interaction will have to be introduced.

Corresponding PM will message to L3 config agent to request a flush when needed. Main usage mode is prior to entering RC6. The rest of the treatment in the nodes is the same. Once flush/invalidate event is complete, message(s) will be sent back to PM with the completion status.

L3 Allocation and Programming

L3 Cache allocation is done on a per way basis which should be consistent across all 4 banks (2 banks for GT1). The way allocation between URB and any of the L3 clients can only be changed post pipeline flush where L3 contains no data. This is required for stream based flushes to be dependent on the way allocation of these corresponding streams. S/W should not be removing ways under a particular stream and expect a later pipelined stream flush to target all the corresponding locations. The stream based flush will be performed on the existing way allocation of that stream, there is no history of previous way allocation tracked in the hardware.

L3 Cache has been divided into following client pools:

- Shared Local Memory: When enabled its size is always fixed to 256 KB (128 KB for GT1)
  - 64 KB in the 4th bank per slice is left unused for GT2/3/4 SKUs.
- URB: Local memory space, provides a flexible allocation on per 8 KB granularity
- DC: Data Cluster Data type
- Inst/State: Both instructions and state allocation are combined
- Constants: Pull constants for EUs
- Textures: texture allocation to back-up L2$

In addition to these sub-groups, a collection of groups are generated to bundle multiple clients under the same allocation set:

- All L3 Clients: DC, Inst/State, Constants & Textures
- Read-Only Clients: Inst/State, Constants & Textures

If pLRU is used, the best performance can be attained via assigning a power-of-2 number of ways. This is to ensure pLRU to distribute the ways without hot spotting within that client's group. Even though design provides a flexible (per way basis) programming model for way allocation for each client following table is given for validation and SW programming models. The programming options in the following table represents most likely cases for different operation modes.

L3 cache requires cache control parameter consistency between the surfaces that they have overlapping sections. If multiple buffers are defined to overlap, the cache policy for L3 is required to be the same between them.

L3 Space allocation can only be changed when the GPU pipeline is completely flushed. To guarantee that following two events need to be executed prior to the inline register updates to L3 allocation registers:

1. PIPECONTROL FLUSH, CS Stall set, with HDC Flush set, RO cache invalidation set if required (This flush command ensures the workload is completely drained, Datapipe is completely flushed followed by initiation of RO cache invalidation. Doesn't ensure RO cache invalidation is complete.)

2. PIPECONTROL FLUSH, CS Stall set, with HDC flush. (This flush ensures any prior RO cache invalidation in progress to be complete before processing flush for this command, this avoids RO cache invalidation colliding with following LRI.)

The following settings are given for GT2.

- GT1 has half of the allocation listed below.
- GT3 has 2x the allocation listed below.
- GT4 has 3x the allocation listed below.

(KBytes)Normal Banked - No SLM									
	SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
0	0	384	384	0	0	0	0	0	768
1	0	384	0	128	256	0	0	0	768
2	0	256	0	128	384	0	0	0	768
3	0	256	0	0	512	0	0	0	768
4	0	256	512	0	0	0	0	0	768
(KBytes)Shared Local Memory Mode - highly banked M									
	SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
1	256	128	384	0	0	0	0	0	768
2	256	128	0	128	256	0	0	0	768
3	256	128	0	256	128	0	0	0	768

Description
URB is limited to 1008KB due to programming restrictions. This is not a restriction of the L3 implementation, but of the FF and other clients. Therefore, in a GT4 implementation it is possible for the programmed allocation of the L3 data array to provide 3*384KB=1152KB for URB, but only 1008KB of this will be used.

Unlike previous generations, logic is simplified via combining RO clients under one pool, given most clients already have their L1 and L2 caches. Bundling them greatly reduces the timing paths and complexity in trying to determine which client belongs to which way.

On Gen9, RW can use the entire 64 ways allocated to L3, and RO can also use the entire 64 ways. In Gen8, RW was restricted to the first 32 ways. So, effectively, the RO and RW memory spaces have been merged into one global space. All ways now have full MESI-state tracking.

Similar to previous generations before changing the configuration of L3, the entire contents needs to be flushed and invalidated. This is to prevent any false hits post configuration update. The invalidation can be achieved either thru command streamer's pipeline flush + top of the pipe invalidations or the global invalidation option within L3. Both options will work for non-coherent option of L3.

For coherent L3 the only option is to follow pipeline invalidation via an L3 global invalidation. That would push all coherent content out of L3 as well allowing a programming update in allocation.

R/W ways allocated (Rest and DC) can not be any size less than 128KB unless they are 0KB.

The locations that are tagged with physical addresses do not need to be flushed between configuration changes.

The following table is added to define the capabilities of programmability in L3.

(KBytes)Normal Banked - No SLM									
SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
0	0 to 512 increments of 16KB	0 to 512 increments of 16KB	0	0	0	0	0	768
0	0 to 512 increments of 16KB	0	0 to 512 increments of 16KB	0 to 512 increments of 16KB	0	0	0	768

(KBytes)Shared Local Memory Mode - highly banked M								
SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
256	0 to 512 increments of 16KB	0 to 512 increments of 16KB	0	0	0	0	0	768
256	0 to 512 increments of 16KB	0	0 to 512 increments of 16KB	0 to 512 increments of 16KB	0	0	0	768

Programming Notes	
Context:	L3 Allocation and Programming

- Allocating the entire cache to DC (Data space) with 0 KB for reads is not allowed. i.e., when Rest is programmed to 0KB, then RO cannot be 0KB
- However, you can program the entire cache as RO or Rest.
- 64KB in SLM space is left unused for GT2/3/4 SKUs.
- A pipeline flush is required prior to changes in L3 allocation and programming. If the L3 contains coherent lines, it must be ensured that these coherent lines are also flushed through the use of the pipe line flush coherent lines setting in L3SQCREG4 prior to the pipeline flush.
- When changing the L3 mode from SLM to non-SLM or vice versa, a CS stalling pipeline flush is required prior to and following the programming of the SLM Mode Enable bit in the L3CNTLREG.
- In the usage model where data is produced by the data port and consumed via the sampler, the cache

Programming Notes	
Context:	L3 Allocation and Programming
allocation shall always be in the unified mode where "Rest" is allocated a non zero value with no allocations for DC and RO ways. Also, the line replacement mode should be set to 1bLRU only.	

Shared Local Memory

Shared local memory (SLM, also known as highly-banked memory) is a portion of L3 which will be dedicated to EUs as a local memory when enabled.

SLM Behavior and Software Usage

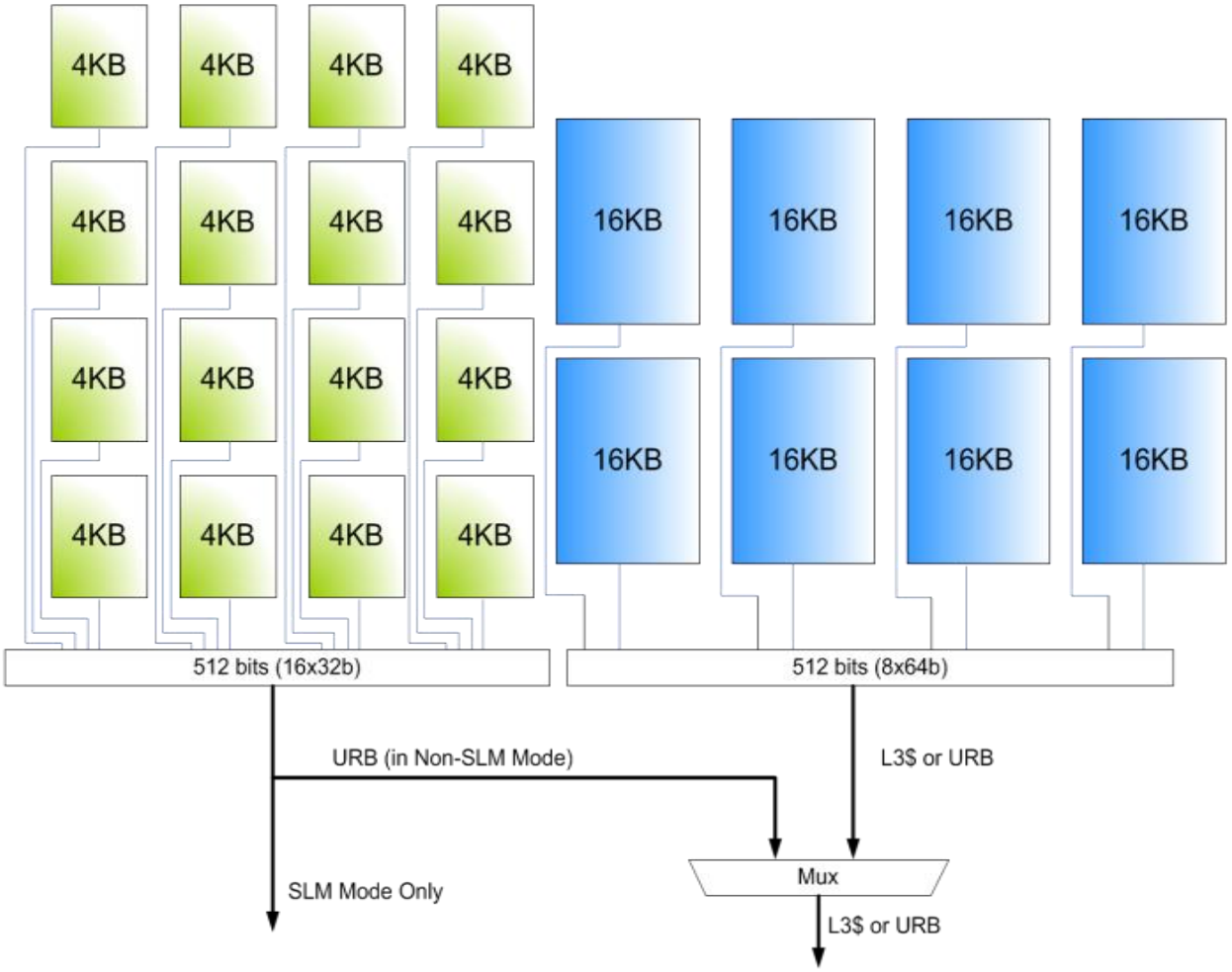
In SKL implementations, the SLM in each physical L3 bank is dedicated to a single sub-slice. All EU's in that sub-slice can access this SLM, but they cannot access the SLM in other L3 banks. SLM is always 64Kbytes in size and is not coherent with system memory. Software is responsible for managing the contents of the SLM.

SLM is enabled statically by the application. When enabled, it uses a fixed 64K block in each L3 Bank. When SLM is enabled, URB will not use this 64K block of memory. URB accesses will automatically be moved to a different location in the L3 bank.

SLM is implemented to allow for highly parallel applications. 16 DWord-accesses (all read or all write) can be done simultaneously to different locations in the 64K space. Atomic operations as described in the Atomic Operation section of this volume can be performed on the output of the SLM and written back in one operation.

SLM Bank Hardware Implementation

The accesses are only possible through data cluster with the destination flag set as SLM. To support a highly banked design, each of the L3 banks are structured to have 16x4KB portion which could be accessed independently per clock. This part of the L3 can support 16 DW size accesses (per SLM) in a given clock cycle. Each bank's SLM is dedicated to a single sub-slice. The diagram below shows the organization of data for a 192KB L3 Bank.

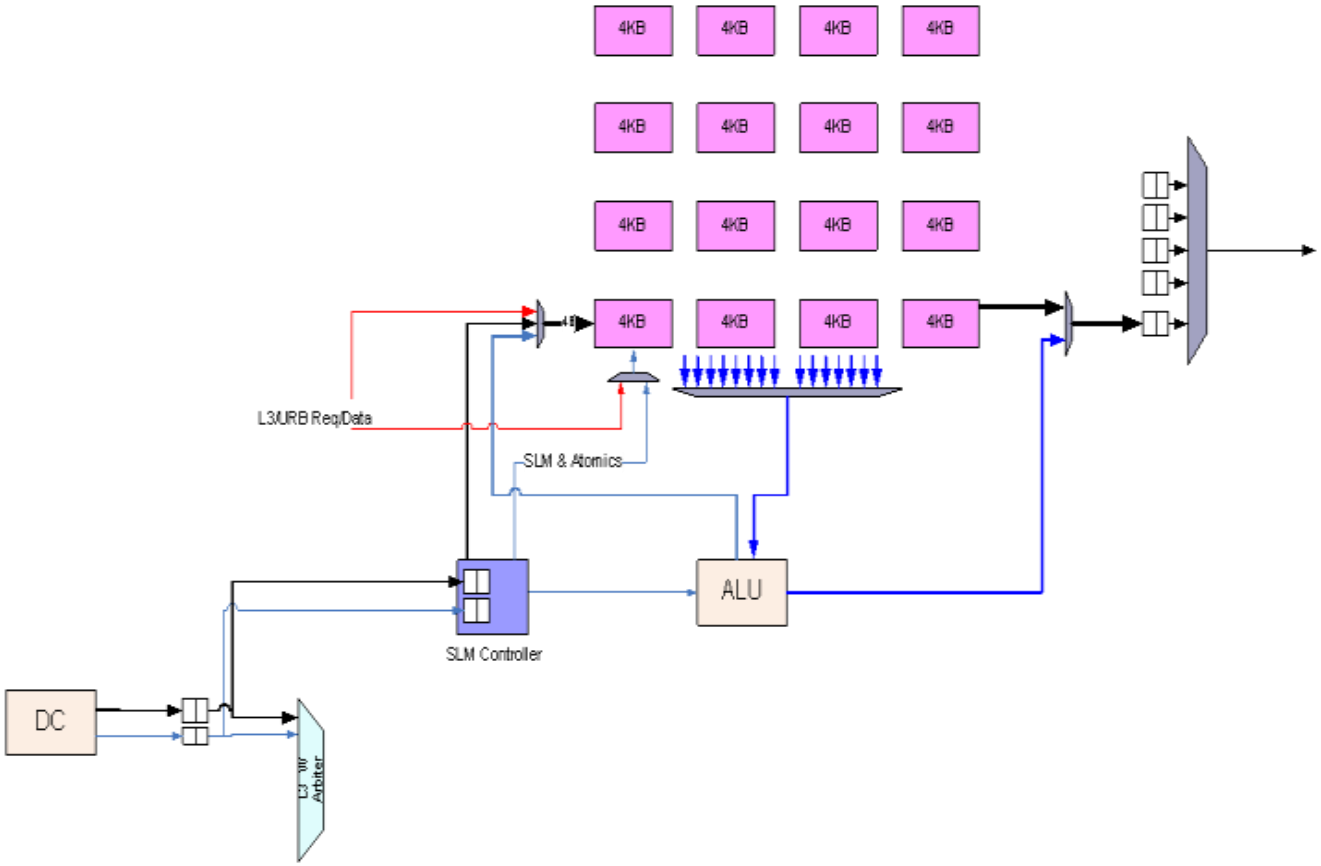


These 16 banks can either be used as URB or used as shared local memory with parallel accesses to all banks. The choice of enabling SLM mode is done through MMIO programming

Bits	Access	Default Value	Description
0	RW/C	0	Enable Shared Local Memory: When set, it enables the use of 2 banks of L3 as shared local memory which allows 64KB of L3 to be banked as 16x4KB and allows independent accesses to all banks within the same clock cycle. **Note:** This mode can only be enabled once L3 content is completely flushed.

SLM is structured as 64KB per L3 bank and each bank allows up to 16 parallel accesses to the 4KB banks. Each 4KB bank is addressed separately where their requests are placed on a 160-bit address bus where each 10-bit correspond to a bank sequentially. Each bank gets addressed with 10 bits and provides 4B access with a total of 4KB per bank. When SLM mode is not enabled, SLM banks are considered a part of

URB.



Save and Restore Requirements

At the pre-emption time for GPGPU the contents of shared local memory will be saved to memory. The flow requires some special handling both in SuperQ and L3 pipeline and will be managed via a small DMA engine which will need to interface to L3 arbiter.

A simple DMA controller within the L3 bank will be designed to generate the accesses that are needed, this block will be used for both save and restore transaction.

Note that GFX virtual buffer given for save and restore has to be 128KB aligned (64KB for GT1).

As part of the process of preemption, the GPGPU controller will ensure there is no more messages from EUs to shared local memory. Once guaranteed, there will be a message to State Arbiter which in turn converted to a message to SLM DMA controller block. From this point on SLM DMA controller will interface with L3 arbiter like any other GFX client with the credit protocol that are defined.

Once all requests are retired to L3 arbiter, the DMA controller will send a completion signal to SARB. State arbiter will wait for both DMA controllers to send their completion signals before sending the response back to GPGPU controller *(note for GT1 – I response is good enough)*

For Save, once the signal is received from State Arbiter to kick-off the save along with the address within the gfx virtual space, DMA controller will start generating following special cycles towards L3 arbiter. Similarly for restore another special request will be sent.

- Save SLM Request
- Restore SLM Request

The request to L3 controller will be 64B size and carry a gfx address similar to any other request that L3 arbiter supports. DMA controller will walk all 64KB address space (1024 entries) via an increment and using the provided address as the base address for request to L3 arbiter.

Once the Save or Restore cycles are all posted to L3 arbiter, DMA controller will signal back to State Arbiter to send the ACK message back to requester.

In L3 arbiter, the save and restore requesters from DMA controller is treated as any other request interface. There is no data involvement so only request queues are needed. A 4 deep structure (ingress FIFO) is provided giving 4 credits to SLM DMA controller to consume.

L3 arbiter will mux in the new ingress FIFO output with HDC read paths and arbitrate Fixed Priority (SLM DMA controller > HDC reads).

The only remaining requirement for L3 arbiter to ensure the flush of this ingress FIFO upon a pipelined Flush received from DC.

In processing save and restore cycles existing flows will be overloaded in SuperQ, both the save and restore request is treated slightly different in SQ FSM.

L3 Cache Error Protection

L3 cache error protection is covered via ECC (secded).

All accesses (including SLM) are subject to ECC protection where single bit errors are fixed silently. Double bit errors are reported via a register structure and communicated by an interrupt to GFX driver. L3 cache HW is additionally capable of stalling execution upon a double bit error; the mechanism is controlled via x7034[9]. If a stall condition is required, the GFX driver is required to initialize the URB prior to execution.

SKL A0 does not have a self-init mechanism for ECC bits between context switches. Lack of self-init shall lead to corruptions due to poisoning of the contents after double bit errors. The w/a that has to be used for A0 stepping is to set: bypass ECC check by programming 0xB120[31]= 1 and 0xB120[0] = 1.

L3 cache and URB have ECC protection on 256b level and SLM has parity based protections. Hardware bases the protection around both URB and SLM to be written before being read, hence there is no automatic initialization. If there is any case where the contents of URB and/or SLM are being read before being written, that could lead to false ECC errors. Hardware does not care about the values read from uninitialized locations, however double-bit ECC errors can trigger an optional "HW lock-up" mode (enabled via MMIO) to protect the contents. Software has to make a choice:

1. For the cases where the workload does not require special double-bit protection, there is no need to enable the "HW lock-up" mode.
2. For the cases where the workload demands an accurate operation and relies on HW detected double-bit errors to lead to HW lock-up, SW will have to initialize the contents of URB or SLM (based on the workload usage model).

L3 Cache and URB

These are the L3 Cache and URB Registers for Gen9. Some of the content is identical to Gen8, with important changes to enhance performance.

LBCF Registers

L3SQCREG1 - L3 SQCRegisters1

L3SQCregisters2

L3SQCregisters3

L3ControlRegister1

L3SLMRegister

L3SQCregister4

SCRATCH1

LTCDconfigregister

LBSconfigbits

L3BankStatus

LBCFconfigsavemsg

LNCF Registers

ArbiterControlRegister

SCRATCHforLNCFunit

Eventselectionandbasecounters

EventSelectionandBaseCounters1

LPFCcontrolregister

LNCFconfigsavemsg

LPFC Registers

PMflushandslmsaverestore

GlobalInvalidationRegister

PowercontextSaveRegisterforLPFC

FirstBufferSizeandStart

SecondBufferSize

ErrorReportingRegister

FramecountandDrawcallnumber

SaveTimer

Masterstarttimer

EU Overview

The GEN instruction set is a general-purpose data-parallel instruction set optimized for graphics and media computations. Support for 3D graphics API (Application Programming Interface) Shader instructions is mostly native, meaning that GEN efficiently executes Shader programs. Depending on Shader program operation modes (for example, a Vertex Shader may be executed on a base of a vertex pair, while a Pixel Shader may be executed on a base of a 16-pixel group), translation from 3D graphics API Shader instruction streams into GEN native instructions may be required. In addition, there are many specific capabilities that accelerate media applications. The following feature list summarizes the GEN instruction set architecture:

- SIMD (single instruction multiple data) instructions. The maximum number of data elements per instruction depends on the data type.
- SIMD parallel arithmetic, vector arithmetic, logical, and SIMD control/branch instructions.
- Instruction level variable-width SIMD execution.
- Conditional SIMD execution via destination mask, predication, and execution mask.
- Instruction compaction.
- An instruction may executed in multiple cycles over a SIMD execution pipeline.
- Most GEN instructions have three operands. Some instructions have additional implied source or destination operands. Some instructions have explicit dual destinations.
- Region-based register addressing.
- Direct or indirect (indexed) register addressing.
- Scalar or vector immediate source operand.
- Higher precision accumulator registers are architecturally visible.
- Self-modifying code is not allowed (instruction streams, including instruction caches, are read-only).

CoIssue/Dual Issue:

The EM Pipe has been expanded to handle general integer and all the float operations that FPU pipe can execute as well.

Thread scheduling:

Threads are scheduled with the "oldest first" policy: a thread runs as long as no dependency is encountered. When a switch is required, the oldest thread i.e., the thread which has been spawned the first is the next to execute. After scheduling the next instruction from the currently executing thread, if any of the four units are free, the EU tries to fill them from instructions from other threads (processed in oldest to newest order).

Introduced an additional thread scheduling policy. This new policy issues the thread in a *round robin* fashion as opposed to *oldest first*. It is a global selection policy meaning that all the EUs are selected to run one policy or the other in an any given time.

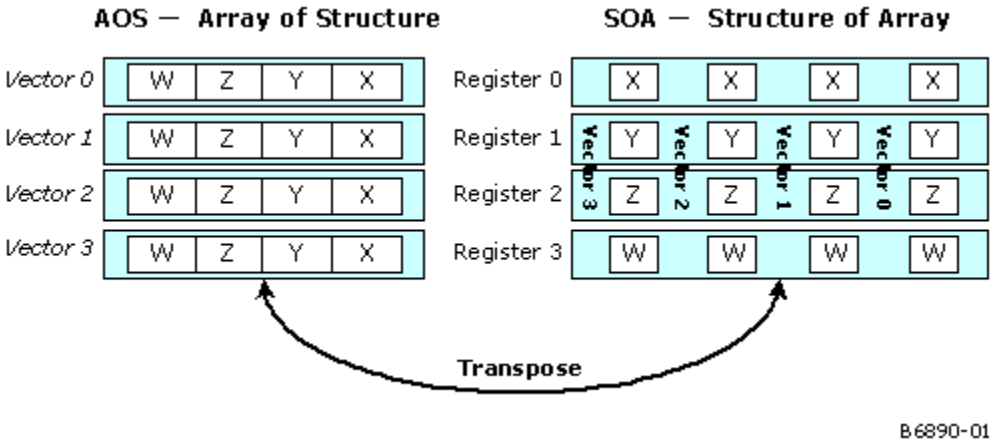
Primary Usage Models

In describing the usage models of the GEN instruction set, the following sections forward reference terminology, syntax, and instructions described later in this specification. For clarity reasons, not all forward references are explained at the point of reference. See the Instruction Set Summary chapter for information about instruction fields and syntax.

AOS and SOA Data Structures

With the Align1 and Align16 access modes, the GEN instruction set provides effective SIMD computation whether data is arranged in array of structures (AOS) form or in structure of arrays (SOA) form. The AOS and SOA data structures are illustrated by the examples in *AOS and SOA Data Structures*. The example shows two different ways of storing four vectors in four SIMD registers. For simplicity, the data vector and the SIMD register both have four data elements. The four data elements in a vector are denoted by X, Y, Z, and W just as for a vertex in 3D geometry. The AOS structure stores one vector in a register and the next vector in another register. The SOA structure stores one data element of each vector in a register and the next element of each vector in the next register and so on. The two structures can be related by a matrix transpose operation.

AOS and SOA Data Structures



GEN 3D and media applications take advantage of such broad architecture support and use both AOS and SOA data arrangements.

- Vertices in 3D Geometry (Vertex Shader and Geometry Shader) are arranged in AOS form and use SIMD4x2 and SIMD4 modes, respectively, as detailed below.
- Pixels in 3D Rasterization (Pixel Shader) are arranged in SOA form and use SIMD8 and SIMD16 modes as detailed below.
- Pixels in media are primarily arranged in SOA form, and occasionally in AOS form with possibly mixed modes of operation that uses region-based addressing extensively.

These are preferred methods; alternative arrangements may also be possible. Shared function resources provide data transpose capability to support both modes of operations: The sampler has a transpose for sample reads, the data port has a transpose for render cache writes, and the URB unit has a transpose for URB writes.

The following 3D graphics API Shader instruction is used in the following sections to illustrate various operation modes:

```
add dst.xyz src0.yxzw src1.zwxy
```

This example is a SIMD instruction that takes two source operands src0 and src1, adds them, and stores the result to the destination operand dst. Each operand contains four floating-point data elements. The data type is determined by the instruction opcode. This instruction also uses source swizzles (.yxzw for src0 and .zwxy for src1) and a destination mask (.xyz). Please refer to the programming specifications of 3D graphics API Shader instructions for more details.

A general register has 256 bits, which can store 8 floating point data elements. For 3D graphics, the mode of operation is (loosely) termed after the data structure as SIMD*mxn*, where *m* is the size of the vector and *n* is the number of concurrent program flows executed in SIMD.

Execution with AOS data structures:

- **SIMD4** (short for SIMD4x1) indicates that a SIMD instruction operates on 4-element vectors storedin registers. There is one program flow.
- **SIMD4x2** indicates that a SIMD instruction operates on a pair of 4-element vectors in registers. There are effectively two programs running side by side with one vector per program.

Execution with SOA data structures, also referred to as "channel serial" execution, mostly uses:

- **SIMD8** (short for SIMD1x8) indicates a SIMD instruction based on the SOA data structure where one register contains one data element (the same one) for each of 8 vectors. Effectively, there are 8 concurrent program flows.
- **SIMD16** (short for SIMD1x16) indicates that a SIMD instruction operates on a pair of registers that contain one data element (the same one) for each of 16 vectors. SIMD16 has 16 concurrent program flows.

SIMD4 Mode of Operation

With a register mapping of src0 to doublewords 0-3 of *r2*, src1 to doublewords 4-7 of *r2* and dst to doublewords 0-3 of *r3*, the example 3D graphics API Shader instruction can be translated into the following GEN instruction:

add (4) r3<4>.xyz:f r2<4>.yzwx:f r2.4<4>.zwxy:f {NoMask}

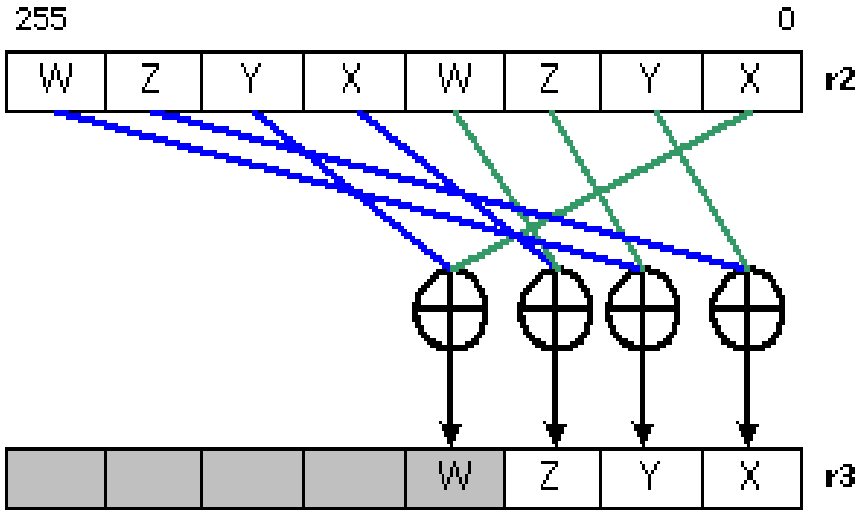
Without diving too much into the syntax definition of a GEN instruction, it is clear that a GEN instruction also takes two source operands and one destination operands. The second term, (4), is the execution size that determines the number of data elements processed by the SIMD instruction. It is similar to the term SIMD Width used in the literature. Each operand is described by the register region parameters such as '<4>' and data type (e.g. "*:f*"). These will be detailed in the SIMD8 Mode of Operation section. The instruction option field, {NoMask}, ensure that the execution occurs for the execution channels shown in the instruction, instead of, possibly, being masked out by the conditional masks of the thread (See Instruction Summary chapter for definition of *MaskCtrl* instruction field).

The operation of this GEN instruction is illustrated in the following figure. In this example, both source operands share the same physical GRF register r2. The two are distinguished by the subregister number. The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 are not modified. In particular, doublewords 4-7 are unchanged as the execution size is 4; doubleword 3 is unchanged due to the destination mask setting.

In this mode of operation, there is only one program flow – any branch decision will be based on a scalar condition and apply to the whole vector of four elements. Option {*NoMask*} ensures that the instruction is not subject to the masks. In fact, most of the instructions in a thread should have {*NoMask*} set.

Even though the execution only performs four parallel add operations, the GEN instruction still executes in 2 cycles (with no useful computation in the second cycle).

A SIMD4 Example



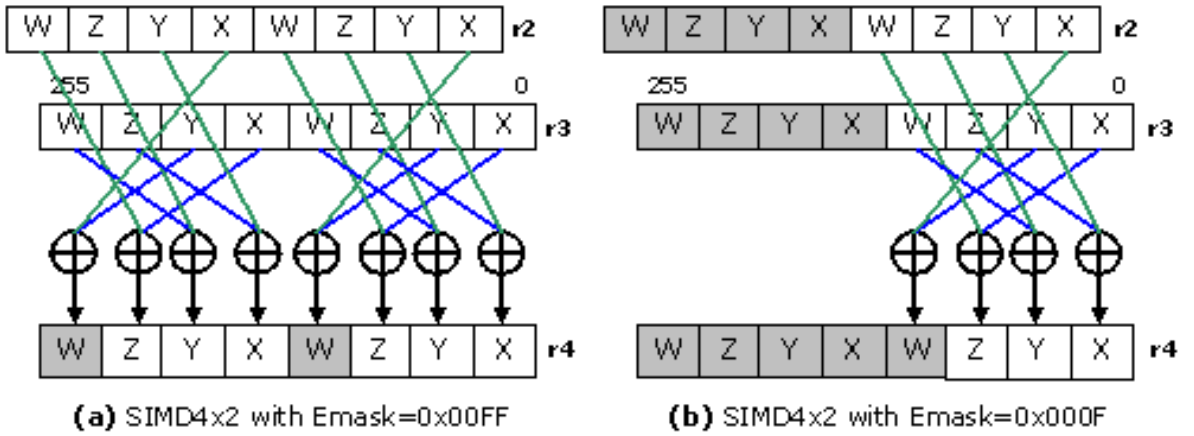
SIMD4x2 Mode of Operation

In this mode, two corresponding vectors from the two program flows fill a GEN register. With a register mapping of src0 to r2, src1 to r3 and dst to r4, the example 3D graphics API Shader instruction can be translated into the following GEN instruction:

add (8) r4<4>.xyz:f r2<4>.yxzw:f r3<4>.zwxy:f

This instruction is subject to the execution mask, which initiated from the dispatch mask. If both program flows are available (e.g. Vertex Shader executed with two active vertices), the dispatch mask is set to 0x00FF. The operation of this GEN instruction is illustrated in *SIMD4x2 Mode of Operation* (a). The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 (doublewords 3 and 7) are unchanged due to the destination mask setting. If only one program flow is available (e.g. the same SIMD4x2 Vertex Shader with only one active vertex), the dispatch mask is set to 0x000F. The operation of the same instruction is shown in *SIMD4x2 Mode of Operation* (b).

SIMD4x2 Examples with Different Emasks

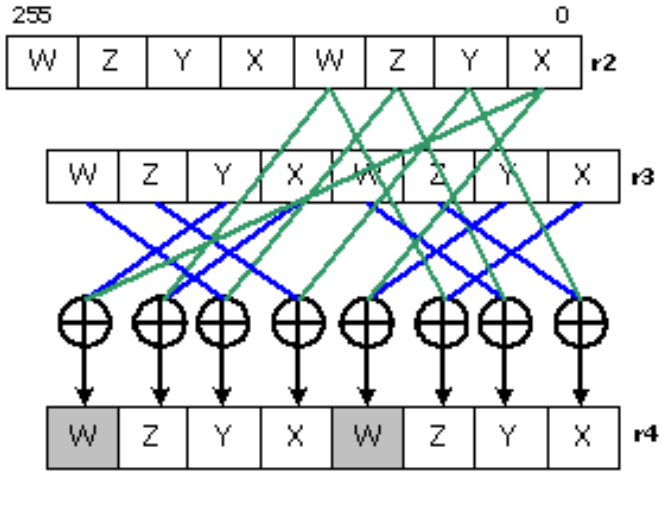


The two source operands only need to be 16-byte aligned, not have to be GRF register aligned. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2, which is shared by the two program flows. The example 3D graphics API Shader instruction can then be translated into the following GEN instruction:

*add (8) r4<4>.xyz:f r2<**0**>.yzwx:f r3<4>.zwxy:f*

The only difference here is that the vertical stride of the first source is 0. The operation of this GEN instruction is illustrated in *SIMD4x2 Mode of Operationicl*.

A SIMD4x2 Example with a Constant Vector Shared by Two Program Flows



SIMD16 Mode of Operation

With 16 concurrent program flows, one element of a vector would take two GRF registers. In this mode, two corresponding vectors from the two program flows fill a GEN register.

With the following register mappings,

src0:r2-r9 (with 16 X data elements in r2-r3, Y in r4-5, Z in r6-7 and W in r8-9),

src1:r10-r17,

dst:r18-r25,

the example 3D graphics API Shader instruction can be translated into the following three GEN instructions:

add (16) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f// dst.x = src0.y + src1.z

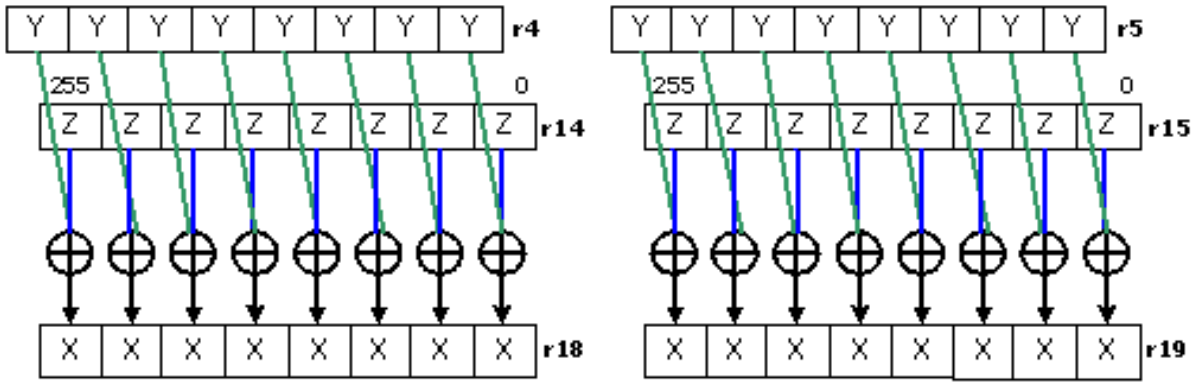
add (16) r20<1>:f r6<8;8,1>:f r16<8;8,1>:f// dst.y = src0.z + src1.w

add (16) r22<1>:f r8<8;8,1>:f r10<8;8,1>:f // dst.z = src0.w + src1.x

The three GEN instructions correspond to the three enabled destination masks As there is no output for the W elements of dst, no instruction is needed for that element. The first instruction inputs the Y elements of src0 and the Z elements of src1 and outputs the X elements of dst. The operation of this instruction is shown in *SIMD16 Mode of Operation*.

With more than one program flow, the above instructions are also subject to the execution mask. The 16-bit dispatch mask is partitioned into four groups with four bits each. For Pixel Shader generated by the Windower, each 4-bit group corresponds to a 2x2 pixel subspan. If a subspan is not valid for a Pixel Shader instance, the corresponding 4-bit group in the dispatch mask is not set. Therefore, the same instructions can be used independent of the number of available subspans without creating bogus data in the subspans that are not valid.

A SIMD16 Example



Similar to SIMD4x2 mode, a constant may also be shared for the 16 program flows. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2 (AOS format). The example 3D graphics API Shader instruction can then be translated into the following GEN instruction:

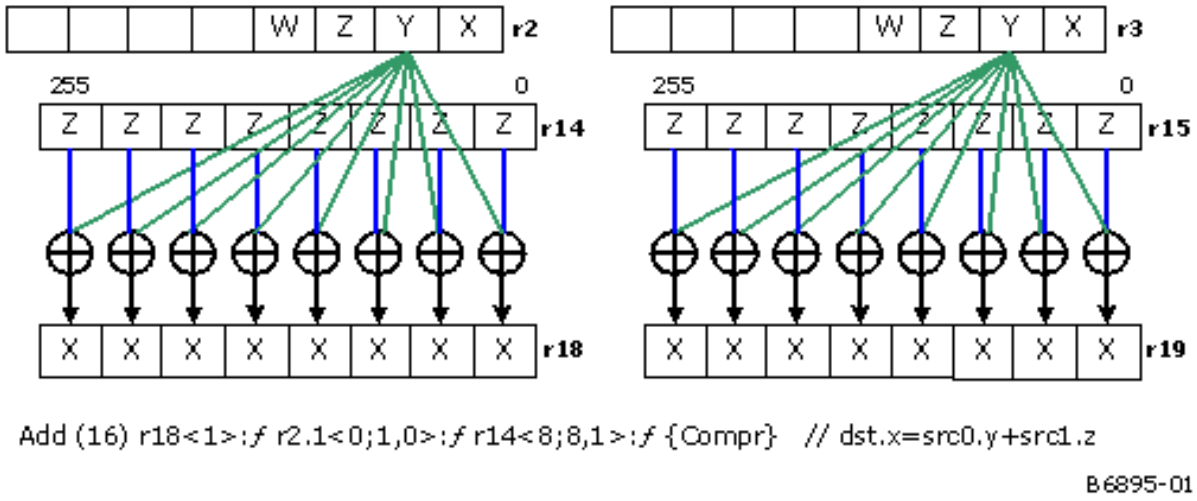
add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr}// dst.x = src0.y + src1.z

add (16) r20<1>:f r2.2<0;1,0>:f r16<8;8,1>:f {Compr}// dst.y = src0.z + src1.w

add (16) r22<1>:f r2.3<0;1,0>:f r10<8;8,1>:f {Compr}// dst.z = src0.w + src1.x

The register region of the first source operand represents a replicated scalar. The operation of the first GEN instruction is illustrated in *SIMD16 Mode of Operation*.

Another SIMD16 Example with an AOS Shared Constant



SIMD8 Mode of Operation

Each compressed instruction has two corresponding native instructions. Taking the example instruction shown in *SIMD16 Mode of Operation*, it is equivalent to the following two instructions.

add (8) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f // dst.x[7:0] = src0.y + src1.z

add (8) r19<1>:f r5<8;8,1>:f r15<8;8,1>:f {SecHalf}

// dst.x[15:8] = src0.y + src1.z

Therefore, SIMD8 can be viewed as a special case for SIMD16.

There are other reasons that SIMD8 instructions may be used. Within a program with 16 concurrent program flows, some time SIMD8 instruction must be used due to architecture restrictions. For example, the address register a0 only have 8 elements, if an indirect GRF addressing is used, SIMD16 instructions are not allowed.

Messages

Communication between the EUs and the shared functions and between the fixed function pipelines (which are not considered part of the "Subsystem") and the EUs is accomplished via packets of information called *messages*. Message transmission is requested via the send instruction. Refer to the send instruction definition in the *ISA Reference* chapter for details.

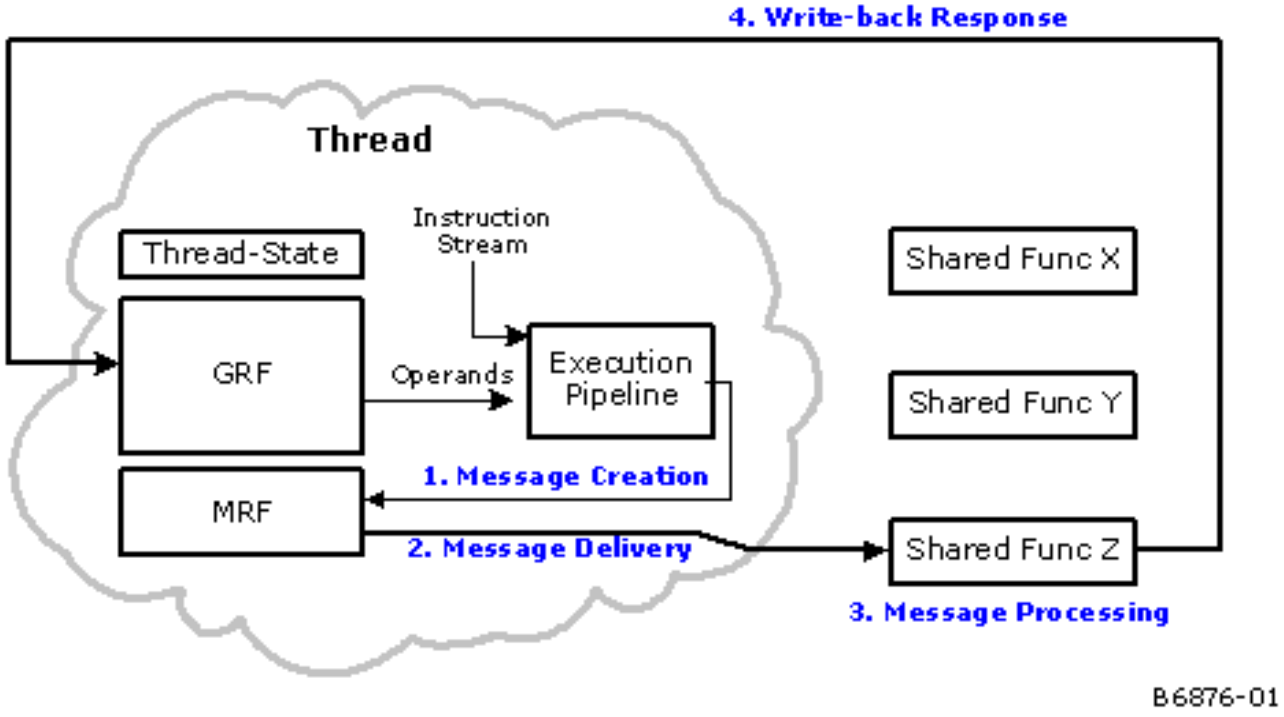
The information transmitted in a message falls into two categories:

- **Message Payload.**
- Associated ("sideband") information provided by:
  - **Message Descriptor.** Specified with the send instruction. Included in the message descriptor is control and routing information such as the target function ID, message payload length, response length, etc.
  - Additional information provided by the send instruction, e.g., the starting destination register number, the execution mask (EMASK), etc.
  - A small subset of Thread State, such as the Thread ID, EUID, etc.

The software view of messages is shown in *Messages* . There are four basic phases to a message's lifetime as illustrated below:

1. **Creation.**
2. **Delivery.** The thread issues the message for delivery via the send instruction. The send instruction also specifies the destination shared function ID (SFID), and where in the GRF any response is to be directed. The messaging subsystem enqueues the message for delivery and eventually routes the message to the specified shared function.
3. **Processing.** The shared function receives the message and services it accordingly, as defined by the shared function definition.
4. **Writeback.** If called for, the shared function delivers an integral number of registers of data to the thread's GRF in response to the message.

Data Flow Associated With Messages



Message Payload Containing a Header

For most shared functions, the first register of the message payload contains the *header payload* of the message (or simply the *message header*). Consequently, the rest of the message payload is referred to as the *data payload*.

Messages to Extended Math do not have a header and only contain data payload. Those messages may be referred to as header-less messages. Messages to Gateway combine the header and data payloads in a single message register.

Writebacks

Some messages generate return data as dictated by the 'function-control' (opcode) field of the 'send' instruction (part of the <desc> field). The Gen4 execution unit and message passing infrastructure do not interpret this field in any way to determine if writeback data is to be expected. Instead explicit fields in the 'send' instruction to the execution unit the starting GRF register and count of returning data. The execution unit uses this information to set in-flight bits on those registers to prevent execution of any instruction which uses them as an operand until the register(s) is(are) eventually written in response to the message. If a message is not expected to return data, the 'send' instruction's writeback destination specifier (<post_dest>) must be set to 'null' and the response length field of <desc> must be 0 (see 'send' instruction for more details).

The writeback data, if called for, arrives as a series of register writes to the GRF at the location specified by the starting GRF register and length as specified in the 'send' instruction. As each register is written

back to the GRF, its in-flight flag is cleared and it becomes available for use as an instruction operand. If a thread was suspended pending return of that register, the dependency is lifted and the thread is allowed to continue execution (assuming no other dependency for that thread remains outstanding).

Message Delivery Ordering Rules

All messages between a thread and an individual shared function are delivered in the ordered they were sent. Messages to different shared functions originating from a single thread may arrive at their respective shared functions out of order.

The writebacks of various messages from the shared functions may return in any order. Further individual destination registers resulting from a single message may return out of order, potentially allowing execution to continue before the entire response has returned (depending on the dependency chain inherent in the thread).

Execution Mask and Messages

The Gen4 Architecture defines an Execution Mask (EMask) for each instruction issued. This 16b bit-field identifies which SIMD computation channels are enabled for that instruction. Since the 'send' instruction is inherently scalar, the EMask is ignored as far as instruction dispatch is concerned. Further the execution size has no impact on the size of the 'send' instruction's implicit move (it is always 1 register regardless of specified execution size).

The 16b EMask is forwarded with the message to the destination shared function to indicate which SIMD channels were enabled at the time of the 'send'. A shared function may interpret or ignore this field as dictated by the functionality it exposes. For instance, the Extended Math shared function observes this field and performs the specified operation only on the operands with enabled channels, while the DataPort writes to the render cache ignore this field completely, instead using the pixel mask included in-band in the message payload to indicate which channels carry valid data.

End-Of-Thread (EOT) Message

The final instruction of all threads must be a *send* instruction that signals 'End-Of-Thread' (EOT). An EOT message is one in which the EOT bit is set in the *send* instruction's 32b <desc> field. When issuing instructions, the EU looks for an EOT message, and when issued, shuts down the thread from further execution and considers the thread completed.

Only a subset of the shared functions can be specified as the target function of an EOT message, as shown in the table below.

Target Shared Functions supporting EOT messages	Target Shared Functions not supporting EOT messages
PixelPort, URB, ThreadSpawner	DataPort, Sampler

Both the fixed-functions and the thread dispatcher require EOT notification at the completion of each thread. The thread dispatcher and fixed functions in the 3D pipeline obtain EOT notification via the target shared functions.

All threads generated by Thread Spawner must send a message to Thread Spawner to explicity signal end-of-thread.

The thread dispatcher, upon detecting an end-of-thread message, updates its accounting of resource usage by that thread, and is free to issue a new thread to take the place of the ended thread. Fixed functions require end-of-thread notification to maintain accounting as to which threads it issued have completed and which remain outstanding, and their associated resources such as URB handles.

Unlike the thread dispatcher, fixed-functions discriminate end-of-thread messages, only acting upon those from threads which they originated, as indicated by the 4b fixed-function ID present in R0 of end-of-thread message payload. This 4b field is attached to the thread at new-thread dispatch time and is placed in its designated field in the R0 contents delivered to the GRF. Thus to satisfy the inclusion of the fixed-function ID, the typical end-of-thread message generally supplies R0 from the GRF as the first register of an end-of-thread message.

As an optimization, an end-of-thread message may be overload upon another "productive" message, saving the cost in execution and bandwidth of a dedicated end-of-thread message. Outside of the end-of-thread message, most threads issue a message just prior to their termination (for instance, a Pixel Port write to the framebuffer) so the overloaded end-of-thread is the common case. The requirement is that the message contains R0 from the GRF (to supply the fixed-function ID), and that destination shared function be either (a) the URB; (b) the Pixel Port; or, (c) the Thread Spawner, as these functions reside on the O-Bus. In the case where the last real message of a thread is to some other shared function, the thread must issue a separate message for the purposes of signaling end-of-thread.

Performance

The Gen4 Architecture imposes no requirement as to a shared function's latency or throughput. Due to this as well as factors such as message queuing, shared bus arbitration, implementation choices in bus bandwidth, and instantaneous demand for that function, the latency in delivering and obtaining a response to a message is non-deterministic. It is expected that a Gen4 implementation has some notion of fairness in transmission and servicing of messages so as to keep latency outliers to a minimum.

Other factors to consider with regard to performance:

Description
Software prefetching techniques may be beneficial for long latency data fetches (i.e. issue a load early in the thread for data that is required late in the thread).

Message Description Syntax

All message formats are defined in terms of DWords (32 bits). The message registers in all cases are 256 bits wide, or 8 DWords. The registers and DWords within the registers are named as follows, where n is the register number, and d is the DWord number from 0 to 7, from the least significant DWord at bits [31:0] within the 256-bit register to the most significant DWord at bits [255:224], respectively. For writeback messages, the register number indicates the offset from the specified starting destination register.

Dispatch Messages: **R**n.d

Dispatch messages are sent by the fixed functions to dispatch threads. See the fixed function chapters in the *3D and Media* volume.

SEND Instruction Messages: **M**n.d

These are the messages initiated by the thread via the SEND instruction to access shared functions. See the chapters on the shared functions later in this volume.

Writeback Messages: **W**n.d

These messages return data from the shared function to the GRF where it can be accessed by thread that initiated the message.

The bits within each DWord are given in the second column in each table.

Message Errors

Messages are constructed via software, and not all possible bit encodings are legal, thus there is the possibility that a message may be sent containing one or more errors in its descriptor or payload contents. There are two points of error detection in the message passing system: (a) the message delivery subsystem is capable of detecting bad FunctionIDs and some cases of bad message lengths; (b) the shared functions contain various error detection mechanisms which identify bad sub-function codes, bad message lengths, and other misc errors. The error detection capabilities are specific to each shared function. The execution unit hardware itself does not perform message validation prior to transmission.

In both cases, information regarding the erroneous message is captured and made visible through MMIO registers, and the driver notified via an interrupt mechanism.

The set of possible errors is listed in *Message Errors* with the associated outcome.

Error Cases

Error	Outcome
Bad Shared Function ID	The message is discarded before reaching any shared function. If the message specified a destination, those registers will be marked as in-flight, and any future usage by the thread of those registers will cause a dependency which will never clear, resulting in a hung thread and eventual time-out.
Unknown opcode	The destination shared function detects unknown opcodes (as specified in the 'send' instructions <desc> field), and known opcodes where the message payload is either too

Error	Outcome
Incorrect message length	long or too short, and threats these cases as errors. When detected, the shared function latches and makes available via MMIO registers the following information: the EU and thread ID which sent the message, the length of the message and expected response, and any relevant portions of the first register (R0) of the message payload. The shared function alerts the driver of an erroneous message through and interrupt mechanism then continues normal operation with the subsequent message.
Bad message contents in payload	Detection of bad data is an implementation decision of the shared function. Not all fields may be checked by the shared function, so an erroneous payload may return bogus data or no data at all. If an erroneous value is detected by the shared function, it is free to discard the message and continue with the subsequent message. If the thread was expecting a response, the destination registers specified in the associated 'send' instruction are never cleared potentially resulting in a hung thread and time-out.
Incorrect response length	Case: too few registers specified – the thread may proceed with execution prior to all the data returning from the shared function, resulting in the thread operating on bad data in the GRF. Case: too many registers specified – the message response does not clear all the registers of the destination. In this case, if the thread references any of the residual registers, it may hand and result in an eventual time-out.
Improper use of End-Of-Thread (EOT)	Any 'send' instruction which specifies EOT must have a 'null' destination register. The EU enforces this and, if detected, will not issue the 'send' instruction, resulting in a hung thread and an eventual time-out. The 'send' instruction specifies that EOT is only recognized if the <desc> field of the instruction is an immediate. Should a thread attempt to end a thread using a <desc> sourced from a register, the EOT bit will not be recognized. In this case, the thread will continue to execute beyond the intended end of thread, resulting in a wide range of error conditions.
Two outstanding messages using overlapping GRF destinations ranges	This is not checked by HW. Due to varying latencies between two messages, and out-of-order, non-contiguous writeback cycles, the outcome in the GRF is indeterminate; may be the result from the first message, or the result from the second message, or a combination of both.

Registers and Register Regions

Register Files

GEN registers are grouped into different name spaces called register files. There are two register files, the General Register File and the Architecture Register File. A third encoding of some register file instruction fields indicates immediate operands within instructions rather than a register file.

- General Register File (GRF): The GRF contains general-purpose read-write registers.
- Architecture Register File (ARF): The ARF contains all architectural registers defined for specific purposes, including address registers (*a#*), accumulators (*acc#*), flags (*f#*), notification count (*n#*), instruction pointer (*ip*), null register (*null*), etc.
- Immediate: Certain instructions can take immediate source operands. A distinct register file field encoding indicates an immediate operand.

Each thread executed in an EU has its own thread context, a dedicated register space that is not shared between threads, whether executing on a common EU or on a different EU. In the rest of the chapters in this volume, register access is relative to a given thread.

GRF Registers

Number of Registers:	Various
Default Value:	None
Normal Access:	RW
Elements:	Various
Element Size:	Various
Element Type:	Various
Access Granularity:	Byte
Write Mask Granularity:	Byte
Indexable?	Yes

Registers in the General Register File are the most commonly used read-write registers. During the execution of a thread, GRF registers are used to store the temporary data, and serve as the destination to receive data from shared function units (and some times from a fixed function unit). They are also used to store the input (initialization) data when a thread is created. By allowing fixed function hardware to initialize some portion of GRF registers during thread dispatch time, GEN architecture can achieve better parallelism. A thread's execution efficiency can also be improved as some data are already in the register to be executed upon. Besides these registers containing thread's payload, the rest of GRF registers of a thread are not initialized.

Summary of GRF Registers

Register File	Register Name	Description
General Register File (GRF)	r#	General purpose read write registers

Each execution unit has a fixed size physical GRF register RAM. The GRF register RAM is shared by all threads on the EU. Each thread has a dedicated space of 128 register, r0 through r127.

GRF registers can be accessed using region-based addressing at byte granularity (both read and write). A source operand must be contained within two adjacent registers. A destination operand must be contained within one register. GRF registers support direct addressing and register-indirect addressing. Register-indirect addressing uses the address registers (ARF registers a#) and an immediate address offset value.

When accessing (read and/or write) outside the GRF register range allocated for a given thread either through direct or indirect addressing, the result is unpredictable.

Register Size	Allocation Granularity	Number per Thread
256 bits	Fixed allocation of 128 registers	128 registers

ARF Registers

ARF Registers Overview

Besides registers that are directly indicated by a unique register file coding, all other registers belong to the Architecture Register File (ARF). Encodings of architecture register types are based on the MSBs of the register number field, RegNum, in the instruction word. The RegNum field has 8 bits. The 4 MSBs, RegNum[7:4], represent the architecture register type. This is summarized in the *Summary of Architecture Registers* table below.

Description
GRF registers are directly indicated by a unique register file encoding.

Summary of Architecture Registers

Register Type (RegNum [7:4])	Register Name	Register Count	Description
0000b	*null*	1	Null register
0001b	*a0.#*	1	Address register
0010b	*acc#*	10	Accumulator register
0011b	*f#.#*	2	Flag register
0100b	*ce#*	1	Channel Enable register
0101b	*msg#*	32	Message Control Register
0110b	*sp*	1	Stack Pointer Register
0111b	*sr0.#*	2	State register

Register Type (RegNum [7:4])	Register Name	Register Count	Description
1000b	*cr0.#*	1	Control register
1001b	*n#*	2	Notification Count register
1010b	*ip*	1	Instruction Pointer register
1011b	*tdr*	1	Thread Dependency register
1100b	*tm0*	2	TimeStamp register
1101b	*fc#.#*	39	Flow Control register
1110b	*Reserved*		Reserved

Programming Note	
Context:	ARF Registers Overview
The remaining register number field RegNum[3:0] is used to identify the register number of a given architecture register type. Therefore, the maximum number of registers for a given architecture register type is limited to 16. The subregister number field, SubRegNum, in the instruction word has 5 bits. It is used to address subregister regions for an architecture register supporting register subdivision. The SubRegNum field is in units of bytes. Therefore, the maximum number of bytes of an architecture register is limited to 32. Depending on the alignment restriction of a register type, only certain encodings of SubRegNum field apply for an architecture register. The detailed definitions are provided in subsequent sections.	

Programming Note	
Context:	ARF Registers Overview
In general an ARF register can be dst (destination) or src0 (source 0, first source operand) for an instruction. Depending on the register and the instruction, other restrictions may apply.	

Access Granularity

ARF registers may be accessed with subregister granularity according to the descriptions below and following the same rule of region-based addressing for GRF.

The machine code for register number and subregister number of ARF follows the same rule as for other register files with byte granularity. For an ARF as a source operand, the region-based address controls the source swizzle mux. The destination subregister number and destination horizontal stride can be used to generate the destination write mask at byte level.

Subregister fields of an ARF register may not all be populated (indicated by the subregister being indicated as reserved). Writes to unpopulated subregisters are dropped; there are no side effect. Reads from unpopulated subregisters, if not specified, return unpredictable data.

Some ARF registers are read-only. Writes to read-only ARF registers are dropped and there are no side effects.

Null Register

Null Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0000b
Number of Registers:	1
Default Value:	N/A
Normal Access:	N/A
Elements:	N/A
Element Size:	N/A
Element Type:	N/A
Access Granularity:	N/A
Write Mask Granularity:	N/A
SecHalf Control?	N/A
Indexable?	No

The null register is a special encoding for an operand that does not have a physical mapping. It is primarily used in instructions to indicate non-existent operands. Writing to the null register has no side effect. Reading from the null register returns an undefined result.

The null register can be used where a source operand is absent. For example, for a single source operand instruction such as MOV or NOT, the second source operand src1 must be a null register.

When the null register is used as the destination operand of an instruction, it indicates the computed results are not stored in any registers. However, implied writes to the accumulator register, if applicable, may still occur for the instruction. When the conditional modifier is present, updates to the selected flag register also occur. In this case, the register region fields of the 'null' operand are valid.

Another example use is to use the null register as the posted destination of a *send* instruction for data write to indicate that no write completion acknowledgement is required. In this case, however, the register region fields are still valid. The null register can also be the first source operand for a send instruction indicating the absent of the implied move. See the *send* instruction for details.

Address Register

Address Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0001b
Number of Registers:	1
Default Value:	None
Normal Access:	RW
Elements:	16

Attribute	Value
Element Size:	16 bits
Element Type:	UW or UD
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	N/A
Indexable?	No

Description
There are sixteen address subregisters forming a 16-element vector. Each address subregister contains 16 bits. Address subregisters can be used as regular source and destination operands, as the indexing addresses for register-indirect-addressed access of GRF registers, and also as the source of the message descriptor for the *send* instruction.

Register and Subregister Numbers for Address Register

RegNum[3:0]	SubRegNum[4:0]
0000b = a0 All other encodings are reserved.	When register a0 or subregisters in a0 are used as the address register for register-indirect addressing, the address subregisters must be accessed as unsigned word integers. Therefore, the subregister number field must also be word-aligned. 00000b = **a0.0:uw** 00010b = **a0.1:uw** 00100b = **a0.2:uw** 00110b = **a0.3:uw** 01000b = **a0.4:uw** 01010b = **a0.5:uw** 01100b = **a0.6:uw** 01110b = **a0.7:uw** 10000b = **a0.8:uw** 10010b = **a0.9:uw** 10100b = **a0.10:uw** 10110b = **a0.11:uw** 11000b = **a0.12:uw** 11010b = **a0.13:uw** 11100b = **a0.14:uw**

RegNum[3:0]	SubRegNum[4:0]
	11110b = **a0.15:uw** All other encodings are reserved. However, when register a0 or subregisters in a0 are explicit source and/or destination registers, other data types are allowed as long as the register region falls in the 128-bit range.

Address Register Fields

DWord	Bits	Description
7	31:16	**Address subregister a0.15:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.14:uw**. Follows the same format as **a0.2**.
6	31:16	**Address subregister a0.13:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.12:uw**. Follows the same format as **a0.2**.
5	31:16	**Address subregister a0.11:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.10:uw**. Follows the same format as **a0.2**.
4	31:16	**Address subregister a0.9:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.8:uw**. Follows the same format as **a0.2**.
3	31:16	**Address subregister a0.7:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.6:uw**. Follows the same format as **a0.2**.
2	31:16	**Address subregister a0.5:uw**. Follows the same format as a0.3.
	15:0	**Address subregister a0.4:uw**. Follows the same format as a0.2.
1	31:16	**Address subregister a0.3:uw**. This field, with only the lower 12 bits populated can be used as an unsigned integer for register-indirect register addressing. Format: U12
	15:0	**Address subregister a0.2:uw**. This field, with only the lower 12 bits populated can be used as an unsigned integer for register-indirect register addressing. Format: U12
0	31:16	**Address subregister a0.1:uw**. This field can be used for register-indirect register addressing or serve as message descriptor for a *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For a *send* instruction, it provides the higher 16 bits of <desc>.

DWord	Bits	Description
		Format: U12 or U16.
	15:0	**Address subregister a0.0:uw**. This field can be used for register-indirect register addressing or serve as message descriptor for a *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For a *send* instruction, it provides the lower 16 bits of <desc>. Format: U12 or U16.

When used as a source or destination operand, the address subregisters can be accessed individually or as a group. In the following example, the first instruction moves 8 address subregisters to the first half of GRF register r1, the second instruction replicates a0.4:uw as an unsigned word to the second half of r1, the third instruction moves the first 4 words in r1 into the first 4 address subregisters, and the fourth instruction replicates r1.4 as a unsigned word to the next 4 address subregisters.

```
mov (8) r1.0<1>:uw a0.0<8;8,1>:uw // r1.n = a0.n for n = 0 to 7 in words
mov (8) r1.8<1>:uw a0.4<0;1,0>:uw // r1.m = a0.4 for m = 8 to 15 in words
mov (4) a0.0<1>:uw r1.0<4;4,1>:uw // a0.n = r1.n for n = 0 to 3 in words
mov (4) a0.4<1>:uw r1.4<0;1,0>:uw // a0.m = r1.4 for m = 4 to 7 in words
```

When used as the register-indirect addressing for GRF registers, the address subregisters can be accessed individually or as a group. When accessed as a group, the address subregisters must be group-aligned. For example, when two address subregisters are used for register indirect addressing, they must be aligned to even address subregisters. In the following example, the first instruction is legal. However, the second one is not. As ExecSize = 8 and the width of src0 is 4, two address subregisters are used as row indices, each pointing to 4 data elements spaced by HorzStride = 1 dword. For the first instruction, the two address subregisters are a0.2:uw and a0.3:uw. The two align to a DWord group in the address register. However, the two address subregisters for the second instruction are a0.3:uw and a0.4:uw. They are not DWord-aligned in the address register and therefore violate the above mentioned alignment rule.

```
mov (8) r1.0<1>:d r[a0.2]<4,1>:d // a0.2 and a0.3 are used for src1
mov (8) r1.0<1>:d r[a0.3]<4,1>:d // ILLEGAL use of register indirect
```

Programming Note	
Context:	ARF Registers
Implementation restriction: When used as the source operand <desc> for the send instruction, only the first dword subregister of a0 register is allowed (i.e. a0.0:ud, which can be viewed as the combination of a0.0:uw and a0.1:uw). In addition, it must be of UD type and in the following form <desc> = a0.0<0;1,0>:ud.	

Programming Note	
Context:	ARF Registers
Restriction: Elements a0.0 and a0.1 have 16 bits each, but the rest of the elements (a0.2:uw through a0.7:uw) only have 12 bits populated each. 12-bit precision supports full indirect-addressing capability for the largest GRF register range. Software must observe the asymmetry of the implementation. When a0.0:uw and a0.1:uw are the source or	

Programming Note	
Context:	ARF Registers
destination, full 16-bit precision is preserved. However, when a0.2:uw to a0.7:uw are the destination, the high 4 bits for each element are dropped; when they are the source, hardware inserts zero to the high 4 bits for each element.	

Programming Note	
Context:	ARF Registers
Performance Note: There is only one scoreboard for the whole address register. When a write to some subregisters is in flight, hardware stalls any instruction writing to other subregisters. Software may use the destination dependency control {NoDDChk, NoDDClr} to improve performance in this case. Similarly, when a write to some subregisters is in flight, hardware stalls any instruction sourcing other subregisters until the write retires.	

Accumulator Registers

Accumulator Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0010b
Number of Registers:	10
Default Value:	None
Normal Access:	RW

Accumulator registers can be accessed either as explicit or implied source and/or destination registers. To a programmer, each accumulator register may contain either 8 DWords or 16 Words of data elements. However, as described in the Implementation Precision Restriction notes below, each data element may have higher precision with added guard bits not indicated by the numeric data type.

Accumulator capabilities vary by data type, not just data size, as described in the Accumulator Channel Precision table below. For example, D and F are both 32-bit data types, but differ in accumulator support.

See the Accumulator Restrictions section for information about additional general accumulator restrictions and also accumulator restrictions for specific instructions.

Accumulator Registers
There are 10 accumulator registers. The accumulator registers are of two types.

Register and Subregister Numbers for Accumulator Registers

RegNum[3:0]	SubRegNum[4]	SubRegNum[3:0]
0000b-1001b = acc0-acc9 All other encodings are reserved.	0 : Lower half 1 : Upper half	Reserved: MBZ

- Accumulators are updated implicitly only if the AccWrCtrl bit is set in the instruction. The Accumulator Disable bit in control register cr0.0 allows software to disable the use of AccWrCtrl for

implicit accumulator updates. The write enable in word granularity for the instruction is used to update the accumulator. Data in disabled channels is not updated.

- When an accumulator register is an implicit source or destination operand, hardware always uses acc0 by default and also uses acc1 if the execution size exceeds the number of elements in acc0. When implicit access to acc1 is required, QtrCtrl is used. Note that QtrCtrl can be used only if acc1 is accessible for a given data type. If acc1 is not accessible for a given data type, QtrCtrl defaults to acc0.

Description
acc0 and acc1 are supported for half-precision (HF, Half Float) and single-precision (F, Float). Use QtrCtrl of Q2 or Q4 to access acc1 for Float. use QtrCtrl of H2 to access acc1 for Half Float.

Examples:

```
// Updates acc0 and acc1 because it is SIMD16:
add (16) r10:f r11:f r12:f {AccWrEn}
// Updates acc0 because it is SIMD8:
add (8) r10:f r11:f r12:f {AccWrEn}
// Updates acc1. Implicit access to acc1 using QtrCtrl:
add (8) r10:f r11:f r12:f {AccWrEn, Q2}
// Updates acc1 for Half Floats using QtrCtrl:
add (16) r10:hf r11:hf r12:hf {AccWrEn, H2}
```

- It is illegal to specify different accumulator registers for source and destination operands in an instruction (e.g. "*add (8) acc1:f acc0:f*"). The result of such an instruction is unpredictable.
-
- Swizzling is not allowed when an accumulator is used as an implicit source or an explicit source in an instruction.
- Reading accumulator content with a datatype different from the previous write will result in undeterministic values.
- Word datatype write to accumulator is not allowed when destination is odd offset strided by 2.
- For any DWord operation, including DWord multiply, accumulator can store up to 8 channels of data, with only acc0 supported.
- When an accumulator register is an explicit destination, it follows the rules of a destination register. If an accumulator is an explicit source operand, its register region must match that of the destination register with the exception(s) described below.

Exceptions

When OWords of accumulators are accessed, the source and destination OWords may be different. For example, the following instructions are allowed:
`mov (4) r10.4<1>:f acc0.0<1>:f`
`add (4) r10.0<1>:f acc0.4<1>:f r11.0<1>:f`
`mov (8) r10.8<1>:uw acc0.0<1>:uw`
`add (8) r10.0<1>:uw acc0.8<1>:uw r11.0<1>:uw`

Exceptions
The source and destination datatypes MUST be the same for such access of accumulator. If destination is contained within one register, source must also be contained within one accumulator register.

Implementation Precision Restriction: As there are only 64 bits per channel in DWord mode (D and UD), it is sufficient to store the multiplication result of two DWord operands as long as the post source modified sources are still within 32 bits. If any one source is type UD and is negated, the negated result becomes 33 bits. The DWord multiplication result is then 65 bits, bigger than the storage capacity of accumulators. Consequently, the results are unpredictable.

Implementation Precision Restriction: As there are only 33 bits per channel in Word mode (W and UW), it is sufficient to store the multiplication result of two Word operands with and without source modifier as the result is up to 33 bits. Integers are stored in accumulator in 2's complement form with bit 32 as the sign bit. As there is no guard bit left, the accumulator can only be sourced once before running into a risk of overflowing. When overflow occurs, only modular addition can generate a correct result. But in this case, conditional flags may be incorrect. When saturation is used, the output is unpredictable. This is also true for other operations that may result in more than 33 bits of data. For example, adding UD (FFFFFFFFh) with D (FFFFFFFFh) results in 1FFFFFFFEh. The sign bit is now at bit 34 and is lost when stored in the accumulator. When it is read out later from the accumulator, it becomes a negative number as bit 32 now becomes the sign bit.

Accumulator Channel Precision

Data Type	Accumulator Number	Number of Channels	Bits Per Channel	Description
DF	acc0	4	64	When accumulator is used for Double Float, it has the exact same precision as any GRF register.
F	acc0/acc1	8	32	When accumulator is used for Float, it has the exact same precision as any GRF register.
HF	acc0/acc1	16	16	When accumulator is used for Half Float, it has the exact same precision as any GRF register.
Q	N/A	N/A	N/A	Not supported data type.
D (UD)	acc0	8	33/64	When the internal execution data type is doubleword integer, each accumulator register contains 8 channels of (extended) doubleword integer values. The data are always stored in accumulator in 2's complement form with 64 bits total regardless of the source data type. This is sufficient to construct the 64-bit D or UD multiplication results using an instruction macro sequence consisting of *mul*, *mach*, and *shr* (or *mov*).
W (UW)	acc0	16	33	When the internal execution data type is word integer, each accumulator register contains 16 channels of (extended) word integer

Data Type	Accumulator Number	Number of Channels	Bits Per Channel	Description
				values. The data are always stored in accumulator in 2's complement form with 33 bits total. This supports single instruction multiplication of two word sources in W and/or UW format.
B (UB)	N/A	N/A	N/A	Not supported data type.

Accumulators
Accumulators acc2-acc9

These are accumulator registers defined for a special purpose. They are used to emulate IEEE-compliant fdiv and sqrt macro operations. The access is different from acc0 and acc1. Each of these accumulator registers are defined as 256-bit registers having 8 DWords. These may be accessed explicitly or implicitly.

- These registers may be accessed explicitly only by a *mov* operation, with no source modifiers, condition modifiers, or saturation. When accessed explicitly, the datatype must be D. On reads, the low 2 bits of each DWord are valid data. The other bits are undefined. On writes, the low two bits are updated and other bits are dropped.

  Example:

```
// Move 256 bits from acc2 to r10. Just low two bits of each DWord are valid:

mov (8) r10:ud acc2:ud

// Move 256 bits from r10 to acc2. Just low two bits of each DWord are updated:
mov (8) acc2:ud r10:ud
```

- These registers are accessed implicitly by three opcodes defined for the macro operations. **Note:** These macro operations are defined under the *math* opcode section. The macro descriptions also define the restrictive implicit uses of these registers.

Description

Implicit access across accumulator registers is required for each source operand for these macro instructions. These opcodes are accessed in Align16 mode only. The Channel Select bits in the instruction are used to implicitly address the different accumulators for each source. Similarly the Channel Enable bits are used to implicitly address the accumulators for destination. The noacc value is specified when no write to accumulator is required; think of it as a null.

Encoding	Accumulator Register
00000001b	acc3
00000010b	acc4
00000011b	acc5

Description	
00000100b	acc6
00000101b	acc7
00000110b	acc8
00000111b	acc9
00001000b	noacc

Flag Register

Flag Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0011b
Number of Registers:	2
Default Value:	None
Normal Access:	RW
Elements:	2
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	Yes
Indexable?	No

Description
There are two flag registers, f0 and f1.

Each flag register contains two 16-bit subregisters. Each flag bit corresponds to a data channel. Predication uses flag values to enable or disable channels. Conditional modifiers assign flag values. If an instruction uses both predication and conditional modifiers, both features use the same flag register or subregisters.

Description
Flags can be split to halfs, quarters, or eighths using the QtrCtrl and NibCtrl instruction fields. Those fields affect the selection of flags for predication and conditional modifiers, but do not affect reading or writing flags as explicit instruction operands.

The values held in the individual bits of a flag register are the result of the most recent instruction with a conditional modifier and specifying that flag register. For example:

```
add.nz.f0.0 ...
```

Updates flag subregister f0.0 with the per-channel results of the not zero condition.

The flag register has per-bit write enables. When being updated as the secondary destination associated with a conditional modifier, only the bits corresponding to the enabled channels in *EMask* are updated. Other bits in the flag subregister are unchanged.

Flag registers and subregisters can also be explicit source or destination operands.

The *sel* instruction does not update flags.

Register and Subregister Numbers for Flag Register

RegNum[3:0]	SubRegNum[4:0]
0000b = f0:ud 0001b = f1:ud Other encodings are reserved.	00000b = f*n*.0:uw 00010b = f*n*.1:uw Other encodings are reserved.

Description
Reference an entire flag register as f0:ud or f1:ud. Reference the flag subregisters as f0.0:uw, f0.1:uw, f1.0:uw, and f1.1:uw.

Channel Enable Register

Channel Enable Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0100b
Number of Registers:	1
Default Value:	N/A
Normal Access:	RO
Elements:	1
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	N/A
SecHalf Control?	No
Indexable?	No

Register and Subregister Numbers for Channel Enable Register

RegNum[3:0]	SubRegNum[4:0]
0000b = ce	00000b = **ce:ud**.

RegNum[3:0]	SubRegNum[4:0]
All other encodings are reserved.	All other encodings are reserved.

Channel Enable Register Fields

DWord	Bits	Description
0	31:0	**Channel Enable Register ce:ud** Format: U32 This field contains 32 bits of Channel Enables for the current instruction.

Message Control Registers

Message Control Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0101b
Number of Registers:	8
Default Value:	None
Normal Access:	RW
Elements:	2
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Word
Write Mask Granularity:	Word

Register and SubRegister Numbers for Message Control Registers

RegNum[3:0]	SubRegNum[4:0]
0000b - 0111 = msg0 - msg7 All other encodings are reserved.	MBZ

These are specific control registers used to track messaging. These may be saved and restored by the kernel only when a thread is in the context save/restore mode. Access of these registers otherwise, will result in undeterministic behaviour.

Each thread has 8 registers. The granularity of access is always one full register, i.e., 256b. These registers must be accessed with a MOV with no predication, src modifiers or conditional modifiers. They MUST be accessed in direct addressing more only. Access mode is ignored when reading/writing these registers.

These registers must be accessed in order, i.e., reads/writes are in order from msg0 to msg7.

Programming Note

Context:	Message Control Registers
Message Control Registers must never be saved or restored	

Stack Pointer Register

Stack Pointer Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0110b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	2
Element Size:	64 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control	No
Indexable	No

The stack pointer register can be accessed as a unsigned DWord integer (or as a QWord on BDW+). It consists of a read-write register, containing the current stack pointer (sp.0), which is relative to the Generate State Base Address, and a read-only stack limit register (sp.1). The stack pointer is inserted into the message header when data is stored into scratch space as a stack. The stack pointer is managed by software. If the stack pointer exceeds the limit or the space allocated, an exception is triggered. See the Stack Pointer Exception in the Exceptions Section. Writes to the stack pointer must use the {Switch} ThreadCtrl option.

Register and Subregister Numbers for SP Register

RegNum[3:0]	SubRegNum[4:0]
0000b = sp All other encodings are reserved.	00000b = **sp.0:uq**. 01000b = **sp.1:uq** All other encodings are reserved.

SP Register Fields

DWord	Bits	Description
0..1	63:48	**Reserved**. MBZ.
	47:0	**sp.0**. Specifies the current stack pointer. This pointer is relative to the General State Base Address. This register is initialized at thread load to the top of the per thread Scratch Space. The register is R/W. *sp = [scratch space pointer] + [scratch space] - 1* Alternatively, this register may be updated by Software to any flat address space. In such cases, the stack is NOT relative to the General State Base Address. Software must ensure that the address is exclusive for the thread.
2..3	63:48	**Reserved**. MBZ.
	47:0	**sp.1**. Specifies the upper limit for the stack pointer. This pointer is relative to the General State Base Address. This register is initialized at thread load to the limit allocated for stack in the state. See the GPGPU Thread Payload description for details. The register is R/W. *sp_limit = [scratch space pointer] + [stack space limit]* Alternatively, this register may be updated by software, similar to the sp register. In such cases, software is responsible for allocating the right thread stack pointer limit.

State Register

State Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0111b
Number of Registers:	2
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Byte
Write Mask Granularity:	N/A
SecHalf Control?	No
Indexable?	No

Register and Subregister Numbers for State Register

RegNum[3:0]	SubRegNum[4:0]
0000b = sr0 All other encodings are reserved.	Valid encoding range: 00000b – 01100b All other encodings are reserved.
0001b = sr1 All other encodings are reserved.	Valid encoding range: 00000b All other encodings are reserved.

State Register Fields

DWord	Bits	Description
0 (sr0.0:ud)	31:28	**Reserved**. MBZ.
	27:24	**FFID** (Fixed Function Identifier). Specifies which fixed function unit generates the current thread. This field is set at thread dispatch and is forwarded on the message bus for all out-going messages from this thread.
	23	**Priority Class**. This field, when set, indicates the thread belongs to the high priority class, which has higher scheduling priority over any thread with this field cleared. The priority field below determines the relative priority within the same priority class. This field is initialized by the thread dispatcher at thread dispatch time and stays unchanged throughout the life span of the thread. This field is forwarded on the message bus to the message bus arbiter for all out-going messages. It serves as a priority hint for the target shared function. See the Shared Function chapters for whether and how a shared function uses this priority hint. 0 = Low priority class. 1 = High priority class.
	22:19	**Reserved**. MBZ.
	18:16	**Priority**. This field is the relative aging priority of the thread. This field indicates the 'age' of this thread relative to other threads within the EU. No two threads in the same EU can have the same priority number (independent of the priority class value). Within the same priority class, an older thread (with a larger priority number) has higher schedule priority over a younger thread. This field is set to zero at a thread's dispatch. During a thread's run time, this field may or may not be incremented when a new thread is dispatched to the same EU. It is only incremented when another thread's priority number is incremented and reaches the same value. For example, if currently there is a thread with priority 0 on an EU, then dispatching a new thread to that EU causes the old thread's priority number to increment to 1. However, if the active thread (assuming for simplicity that there is only one) on an EU has a priority number 1 (or 2 or 3), then dispatching a new thread to this EU does not change

DWord	Bits	Description
		the old thread's priority number. As threads on an EU may terminate in arbitrary order, the exact number for a thread depends on the dynamic execution of threads. When thread context is saved and restored after pre-emption, the Priority is not restored to the original state. Instead the priority is initiated as if new threads were loaded.
	7:3	**Reserved**. MBZ.
	2:0	**TID** (The thread identifier). Specifies the thread slot that the current thread is assigned to. This field is set at thread dispatch.
2 (sr0.2:ud)	31:0	**Dispatch Mask** (DMask). This 32-bit field specifies which channels are active at Dispatch time. This field is used by hardware to initialize the mask register. Format: U32
3 (sr0.3:ud)	31:0	**Vector Mask** (VMask). This 32-bit field contains, for each 4-bit group, the OR of the corresponding 4-bit group in the dispatch mask. This field is used by hardware to initialize the mask register. Format: U32
0 (sr1.0:ud)	31:0	**Hardware Defined State Register.** The contents of these register are hardware defined and are required only for handling page-fault. These bits are saved and restored by SIP when threads are pre-empted. Writes to these registers must follow the sequence described in 'send' instruction for the correct behavior of hardware.
1 (sr1.1:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0
2 (sr1.2:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0
3 (sr1.3:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0

Implementation Restriction on Register Access: When the state register is used as a source and/or destination, hardware does not ensure execution pipeline coherency. Software must set the thread control field to *'switch'* for an instruction that uses state register as an explicit operand. This is important as the state register is an implicit source or destination for many instructions. For example, fields like IEEE Exception may be an implicit destination updated by multiple back to back instructions. Therefore, if the instructions updating the state register doesn't set *'switch'*, subsequent instructions may have undefined results.

Control Register

Control Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1000b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

The Control register is a read-write register. It contains four 32-bit subregisters that can be accessed individually.

Subregister *cr0.0:ud* contains normal operation control fields such as the floating-point mode and the accumulator disable. It also contains the master exception status/control field that allows software to switch back to the application thread from the System Routine.

Subregister *cr0.1:ud* contains the mask and status/control fields for all exceptions. The exception fields are arranged in significance-decreasing order from MSB to LSB. This arrangement allows the System Routine to use the *lzd* instruction to find the high priority exceptions and handle them first. As each exception is mapped to a single bit, another exception priority order may be implemented by software. The System Routine may choose to handle one exception at a time, by handling the exception detected by an *lzd*instruction and returning to the application thread. Or it may choose to handle all the concurrent exceptions, by looping through the exception fields until all outstanding exceptions are handled before returning back to the application thread.

Exception enable bits (bits 15:0 in *cr0.1:ud*) control whether an exception causes hardware to jump to the System Routine or not. Exception status and control bits (bits 31:16 in *cr0.1:ud*) indicate which exceptions have occurred, and are used by the system routine to clear the exception. Even if a given exception is disabled, the corresponding exception status and control bit still reflects its status, whether an exception event has occurred or not.

cr0.2:ud contains the **Application IP (AIP)** indicating the current thread IP when an exception occurs.

cr0.3:ud is reserved. Values written to this subregister are dropped; the result of reading from this subregister is unpredictable.

Fields in Control registers also reference a virtual register called **System IP (SIP)**. SIP is the virtual register holding the global System IP, which is the initial instruction pointer for the System Routine. The SIP is a

GraphicsAddress. There is only one SIP for the whole system. It is virtual only from a thread's point of view, as it is not visible (i.e. not readable and not writeable) to the thread software executed on a GEN EU. It can only be accessed indirectly by the hardware to respond to exception events. Upon an exception, hardware performs some bookkeeping (e.g. saving the current IP into AIP) and then jumps to SIP. Upon finishing exception handling, the System Routine may return back to the application by clearing the Master Exception Status and Control field in *cr0*, which causes the hardware to load AIP to IP register. See the STATE_SIP command for how to set SIP.

Although the SIP may be more than 32 bits wide, the EU still only uses the low 32 bits.

Register and Subregister Numbers for Control Register

RegNum[3:0]	SubRegNum[4:0]
0000b = cr0 All other encodings are reserved.	00000b = **cr0.0:ud**. It contains general thread control fields. 00100b = **cr0.1:ud**. It contains exception status and control. 01000b = **cr0.2:ud**. It contains AIP. All other encodings are reserved.

Control Register Fields

DWord	Bits	Description
0	31	**Master Exception State and Control.** This bit is the master state and control for all exceptions. Reading a 0 indicates that the thread is in normal operation state and a 1 means the thread is in exception handle state. Upon an exception event, hardware sets this bit to 1 and switches to SIP. Writing 1 to this bit has no effect. Writing 0 to this bit also has no effect if the previous value is 0. In both cases, the bit keeps the previous value. If the previous value of this bit is 1, software writing a 0 causes the thread to return to AIP. This transition is automatic – software does not have to move AIP to IP. The value of this bit then stays as 0. This bit is initialized to 0. 0 = The thread is in normal state. 1 = The thread is in exception state.
	30:16	**Reserved.** MBZ.
	14:13	**Reserved.** MBZ.
	12	**IEEE Float to Integer Rounding Mode.** This bit determines how rounding modes are handled in float to integer conversion operation. This bit is initialized to 0 during Thread Dispatch. This bit must be set for IEEE compliant float to integer conversion operation. 0= The result of float to integer conversion operation is with RTZ rounding mode. 1= The result of float to integer conversion operation is with rounding mode programmed in cr0.0[5:4].

DWord	Bits	Description
	11	**IEEE MinMax.** This bit determines how SNAN is handled in min/max operations. This bit is initialized to 0 during Thread Dispatch. This bit must be set for IEEE compliant min/max operation. 0 = The result of min/max is a non-SNAN source. 1 = The result of min/max is the SNAN source.
	10	**Half Precision Denorm Mode.** This bit determines how denormal numbers are handled for the HF (Half Float) type. This bit is initialized to 0 during Thread Dispatch. 0 = Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign. 1 = Allow denorm source values and denorm results.
	9	**IEEE Exception Trap Enable.** This bit enables trapping IEEE exception flags. This control bit may updated by software. It is initially zero on thread load. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exceptions field of sr0.1. Note that IEEE floating-point exceptions do *not* transfer control to any handler. 0 = IEEE Exception flags are NOT trapped. 1 = IEEE Exception flags are trapped.
	7	**Single Precision Denorm Mode.** This bit determines how denormal numbers are handled for the F (Float) type when using the IEEE floating-point mode. It is ignored in the ALT floating-point mode, which always flushs denorms. This bit is initialized by Thread Dispatch. 0 = Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign. 1 = Allow denorm source values and denorm results.
	6	**Double Precision Denorm Mode.** This bit determines how denormal numbers are handled for the DF (Double Float) type. It is initialized by Thread Dispatch. 0 = Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign. 1 = Allow denorm source values and denorm results.
	5:4	**Rounding Mode.** This field specifies the FPU rounding mode. It is initialized by Thread Dispatch. 00b = Round to Nearest or Even (RTNE) 01b = Round Up, toward +inf (RU) 10b = Round Down, toward -inf (RD) 11b = Round Toward Zero (RTZ)

DWord	Bits	Description
	3	**Vector Mask Enable (VME).** This bit indicates DMask or Vmask should be used by EU for execution. This bit is set by the Thread Dispatch. 0: Use Dispatch Mask (DMASK) 1: Use Vector Mask (VMASK)
	2	**Single Program Flow (SPF).** Specifies whether the thread has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). This bit affects the operation of all branch instructions. In Single Program Flow mode, all execution channels branch and/or loop identically. This bit is initialized by the Thread Dispatch. 0: Multiple Program Flows 1: Single Program Flow **Programming Restrictions:** Only H1/Q1/N1 are allowed in SPF mode. Power Optimization: If an entire shader does not do SIMD branching, the driver can set the SPF bit to 1 to save power in HW.
	1	**Accumulator Disable.** This bit controls the update of the accumulator by the instruction field AccWrCtrl. If this bit is cleared, the accumulator is updated for all instructions with AccWrCtrl enabled. If set, the accumulator is disabled for all update operations, maintaining its value prior to being disabled. Setting this bit has no effect if the accumulator is the explicit destination operand for an instruction. This bit is initialized to 0. 0: Enable accumulator update. 1: Disable accumulator update. **Usage Notes:** This control bit is primarily designed for the System Routine. That routine is not expected to use the accumulator, though it may need to use instructions that implicitly update the accumulator. To use such instructions in the System Routine, but still preserve the accumulator contents on returning to the application kernel, the System Routine would either (a) save and restore the accumulator, or (b) prevent the accumulator from being unintentionally modified. This control bit has been added for the latter method. Software has the option to limit the setting of this control bit to strictly within the System Routine. If, by convention, this bit is clear within application kernels, the System Routine can simply set the bit upon entry and clear it before returning control to the application kernel. This usage model would not necessarily require cr0.0 to be saved/restored in the System Routine. However, if by convention application kernels are permitted to set this bit, then the System Routine is required to preserve the content of this bit.

DWord	Bits	Description
	0	**Single Precision Floating Point Mode (FP Mode).** This bit specifies whether the current single-precision floating-point operation mode is IEEE mode (IEEE Standard 754) or the ALT (alternative mode). This bit does not affect the floating-point mode used for other floating-point data types. This bit is also forwarded on the message sideband for all out-going messages, for example, to control the floating-point mode of the Sampler. Software may modify this bit to dynamically switch between the two floating-point modes. This bit is initialized by Thread Dispatch. 0 = IEEE floating-point mode for the F (Float) type. 1 = ALT (alternative) floating-point mode for the F (Float) type.
1	30	**External Halt Exception Status and Control**. This bit indicates the External Halt exception. It is set by EU hardware on receiving the broadcast External Halt signal. The System Routine should reset this bit before returning to an application routine to avoid infinite loops. This bit may be set or cleared by software. This bit is initialized to 0.
	29	**Software Exception Control**. This bit is the control bit for software exceptions. Setting this bit to 1 in an application routine causes an exception. Clearing this bit in an application routine has no effect. Upon entering the system routine, the hardware maintains this bit as 1 to signify a software exception. The System Routine should reset this bit before returning to an application routine. This bit may be set or cleared by software. This bit is initialized to 0.
	28	**Illegal Opcode Exception Status**. This bit, when set, indicates an illegal opcode exception. The exception handler routine normally does not return back to the application thread upon an illegal opcode exception. Leaving this bit set has no effect on hardware; if system software adversely returns to an application routine leaving this bit set, it doesn't cause any exception. This bit should not be set by software or left set by the system routine to avoid confusion. This bit is initialized to 0.
	27	**Stack Overflow Exception Status**. This bit when set, indicates a stack overflow exception. The exception handler routine normally does not return back to the application thread upon a stack overflow exception. Leaving this bit set has no effect on hardware; if system software adversely returns to an application routine leaving this bit set, it doesn't cause any exception. This bit should not be set by software or left set by the system routine to avoid confusion. This bit is initialized to 0.
	25	**Context Save Status.** This bit when set, indicates a Context Save process has been initiated. The system routine must reset this bit after saving the context to terminate the thread.
	24	**Context Restore Status.** This bit when set, indicates a Context Restore process has been initiated. The system routine must reset this bit after restoring the context. The reset of this bit is required before invoking application routine.
	23:16	**Reserved**. MBZ.

DWord	Bits	Description
	13	**Software Exception Enable**. This bit enables or disables the software exception. Enabling or disabling this bit may allow host software to turn on/off certain features (such as profiling) without changing the kernel program. This bit is initialized by the Thread Dispatcher. Format = ENABLED: 0: Disabled 1: Enabled
	12	**Illegal Opcode Exception Enable**. This bit specifies whether the illegal opcode exception is enabled or not. The Illegal opcode exception includes illegal opcodes and undefined opcodes, caused by bad programs or run-time data corruption. This bit is initialized by the Thread Dispatcher. Software should normally assign this bit in the interface descriptor. Even though this mechanism is provided to disable the illegal opcode exception, it should be used with extreme caution. Format = ENABLED: 0: Disabled 1: Enabled
	11	**Stack Overflow Exception Enable**. This bit specifies whether the stack overflow exception is enabled or not. The stack overflow exception includes an overflow or an underflow in the stack space allocated for the thread. This bit is initialized by the Thread Dispatcher. Software should normally assign this bit in the interface descriptor. Format = ENABLED: 0: Disabled 1: Enabled
	10:0	**Reserved.** MBZ.
2 (cr0.2:ud)	31:3	**Application IP (AIP)**. This is the register storing the instruction pointer before an exception is handled. Upon an exception, hardware automatically saves the current IP into the AIP register, and then sets the **Master Exception State and Control** field to 1, which forces a switch to the System IP (SIP). The AIP register may contain either the pointer to the instruction that causes the exception (such as breakpoint) or the one after (such as masked stack overflow/underflow exceptions). This is shown in the following table, where IP is the instruction that generated the exception.

Exception Type	AIP Value
External Halt	N/A (1)

DWord	Bits	Description
		<table><tr><td>Software Exception</td><td>IP + 1</td></tr><tr><td>Illegal Opcode</td><td>IP</td></tr></table> (1) External Halt exception is asynchronous and not associated with an instruction. When the System Routine changes the Master Exception State and Control field from 1 to 0, hardware restores IP from this register. This field is writable allowing the returning IP to be altered after an exception is handled.
	2:0	**Reserved.** MBZ.

Programming Note	
Context:	
Implementation Restriction on Register Access: When the control register is used as an explicit source and/or destination, hardware does not ensure execution pipeline coherency. Software must set the thread control field to *'switch'* for an instruction that uses control register as an explicit operand. This is important as the control register is an implicit source for most instructions. For example, fields like FPMode and Accumulator Disable control the arithmetic and/or logic instructions. Therefore, if the instruction updating the control register doesn't set *'switch'*, subsequent instructions may have undefined results.	

Notification Registers

Notification Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1001b
Number of Registers:	3
Default Value:	No
Normal Access:	RO (RW – Context save/restore only)
Elements:	3
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

There are three notification registers (*n0.0:ud*, *n0.1:ud, and n0.2:ud*) used by the *wait* instruction. These registers are read-only, except under context restore, and can be accessed in 32-bit granularity. Write access to this register is allowed only when context is restored.

Workaround	
Context:	Notification Registers
The sub-register numbers for n0.0 and n0.2 are swapped on a write, i.e., a destination of n0.0 is required to update n0.2 and n0.2 is required to update n0.0.	

It should be noted that in the extreme case, it is possible to have more notifications to a thread than the maximum allowed number of concurrent threads in the system. Therefore, the range of the thread-to-thread notification count in n0, is larger than the maximum number of threads computed by EUID * TID.

There is only one bit for the host-to-thread notification count in n1.

When directly accessed, this register is read-only. If the value is non zero, the only way to alter the value is to use the wait instruction to decrement the value until zero is reached. A wait instruction on a zero notification subregister causes the thread to stall, waiting for a notification signal from outside targeting the same subregister. See the wait instruction for details.

Implementation Restriction: The notification registers are initialized to 0 after hardware/software reset. However, these registers are not reset at thread dispatch time.

Register and Subregister Numbers for Notification Registers

RegNum[3:0]	SubRegNum[4:0]
0000b = n0 All other encodings are reserved.	00000b = n0.0:ud 00100b = n0.1:ud 01000b = n0.2:ud All other encodings are reserved.

Notification Register 0 Fields

DWord	Bits	Description
0	31:16	**Reserved**. MBZ.
	15:0	**Thread to Thread Notification Count**. This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See the WAIT instruction for details. Format: U16

Notification Register 1 Fields

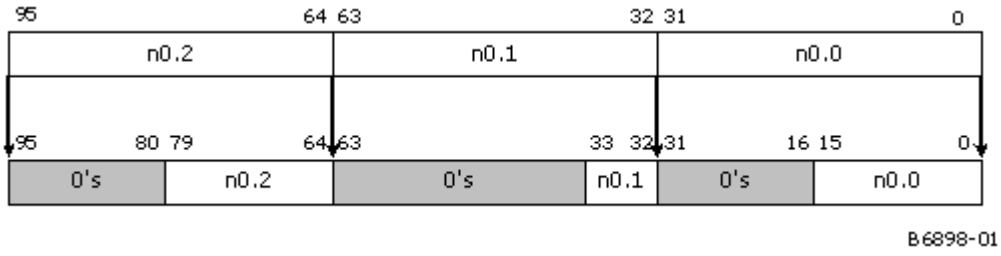
DWord	Bits	Description
0	31:1	**Reserved**. MBZ.
	0	**Host to Thread Notification**. This register is used by the WAIT instruction for host-to-thread

DWord	Bits	Description
		synchronization via MMIO registers. Format: U1

Notification Register 2 Fields

DWord	Bits	Description
0	31:16	**Reserved**. MBZ.
	15:0	**Thread to Thread Notification Count**. This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See the WAIT instruction for details. Format: U16

Format of the Notification Register



IP Register

IP Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1010b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	1
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

The ip register can be accessed as a 32-bit quantity. It is a read-write register, containing the current instruction pointer, which is relative to the **Generate State Base Address**. Reading this register returns the instruction pointer of the current instruction. The 3 LSBs are always read as zero. Writing this register causes program flow to jump to the new address. Writes to this register should use the Switch ThreadCtrl option. When it is written, the 3 LSBs are dropped by hardware.

IP Register Workaround

Register and Subregister Numbers for IP Register

RegNum[3:0]	SubRegNum[4:0]
0000b = ip All other encodings are reserved.	00000b = **ip:ud** All other encodings are reserved.

IP Register Fields

DWord	Bits	Subfield Description
0	31:3	**Ip**. Specifies the current instruction pointer. This pointer is relative to the **General State Base Address**.
	2:0	**Reserved**. MBZ.

TDR Registers

TDR Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1011b
Number of Registers:	1
Default Value:	No
Normal Access:	RO/CW
Elements:	8
Element Size:	16 bits
Element Type:	UW
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	No
Indexable?	No

There are 8 thread dependency registers (tdr0.0:uw to tdr0.7:uw) used by HW for the *sendc* instruction. These registers are read-only and can be accessed in 16-bit granularity.

When accessed explicitly, each thread dependency register has FFTID in the lower 8 bits, bits 8 to 14 are

forced to zero by HW. Bit 15 is the valid bit, which indicate whether the current thread has a dependency on the dependency thread stored in this thread dependency register.

The thread dependency registers are read only, the valids can only be set with a thread dispatch, and are reset by broadcasting end of thread messages after a thread retired. The FFTID's can only be changed with a thread dispatch. Any write into any of the TDR registers will clear the valid bit for the particular TDR if the write enable is true, the FFTID portion is strictly read only.

Register and Subregister Numbers for TDR Registers

RegNum[3:0]	SubRegNum[4:0]
1011b = tdr0 All other encodings are reserved.	00000b = **tdr0.0:uw** 00010b = **tdr0.1:uw** 00100b = **tdr0.2:uw** 00110b = **tdr0.3:uw** 01000b = **tdr0.4:uw** 01010b = **tdr0.5:uw** 01100b = **tdr0.6:uw** 01110b = **tdr0.7:uw** All other encodings are reserved.

TDR Registers Fields

DWord	Bits	Description
3	31	**Valid7**. This field indicates whether the thread specified by FFTID7 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID7**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid6**. This field indicates whether the thread specified by FFTID6 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID6**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
2	31	**Valid5**. This field indicates whether the thread specified by FFTID5 is still in-flight.
	30:25	**Reserved**. MBZ

DWord	Bits	Description
	24:16	**FFTID5**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid4**. This field indicates whether the thread specified by FFTID4 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID4**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
1	31	**Valid3**. This field indicates whether the thread specified by FFTID3 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID3**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid2**. This field indicates whether the thread specified by FFTID2 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID2**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
0	31	**Valid1**. This field indicates whether the thread specified by FFTID1 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID1**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid0**. This field indicates whether the thread specified by FFTID0 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID0**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9

Performance Registers

Performance Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1100b
Number of Registers:	1
Default Value:	0h
Normal Access:	RO/RW
Elements:	4
Element Size:	32 bits
Element Type:	UD

Timestamp Register

This register is a low latency timestamp source, "TM", available as part of a thread's Architectural Register File (ARF). This is a is free running counter, 64b in size, and exposed to the ISA as individual 32b high 'TmHigh' and low 'TmLow' unsigned integer source operands. As part of the EU's register space, access to the timestamp has a low and deterministic latency and therefore can be used for intra-kernel high resolution performance profiling.

The TM features provides a 1-bit indicator 'TmEvent' which identifies the occurrence of a time-impacting event such as context switch or frequency change since the last time any portion of the Timestamp register value was read by that thread. Software that uses the Timestamp capability should check this bit to identify when a relative time calculation may be suspect. To properly use this additional information, the instrumentation code should operate on the Timestamp register value as a whole (i.e. as an 8 dword register) so that the 64b time and this 1b value are captured simultaneously, as opposed to 32b portions, to eliminate the chance of missing a TmEvent that might occur between accesses to 32b portions of this register.

Programming Note	
Context:	Performance Registers
The Timestamp register is saved as part of thread state on context-save, but only 'TmEvent' is restored (and technically always restored to '1' as a context switch had occurred).	

Performance Counter Register

This is a counter intended to provide finer grained visibility into the EUs performance inside kernels. This counter is a 32-bit free-running counter that increments if the EU flexible performance event selected for OA counter A7 is true (please refer to OA documentation for details on how to count various EU flexible events on OA counter A7). The pm0 count continues to increment during a thread's active/standby state transitions as well as context switches. It is read-only and not pre- or resettable under any software control, either kernel or driver, other than a full gfx reset.

Register and Subregister Numbers for Performance Register

RegNum[3:0]	SubRegNum[4:0]
0000b = *tm0* All other encodings are reserved.	00000b = **tm0.0:ud.** 00100b = **tm0.1:ud.** 01000b = **tm0.2:ud** 01100b = **tm0.3:ud** 10000b = **tm0.4:ud** **All other encodings are reserved.**

Performance Register Fields

DWord	Bits	Description
0 (tm0.0:ud)	31:0	**TmLow.** The lower 32b of the 64b timestamp value sourced from Cr clock. Read-only. Format: U32
1 tm0.1:ud	31:0	**TmHigh.** The upper 32b of the 64b timestamp value sourced from Cr clock. Read-only. Format: U32
2 tm0.2:ud	31:1	**Reserved**
	0	**TmEvent.** Indicates a discontinuous time-impacting event (e.g. context switch, frequency change) occurred since any portion of the Timestamp register was last read, thus making any relative duration calculation based on this counter suspect. This bit is reset at the time a new thread is loaded, and on each read of any portion of the 'Timestamp' register.
3 tm0.3 (pm0)	31:0	pm0. Increments based on the EU flexible performance event currently selected being true. Format: U32

Flow Control Registers

Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1101b
Number of Registers:	39
Default Value:	None
Normal Access:	RW*

Register and Subregister Numbers for Flow Conrol Registers

RegNum[3:0]	SubRegNum[4:0]
0000b = fc0	00000b-11111b = **fc0.0–fc0.31**.
0001b = fc1	00000b = **fc1.0**. All other encodings are reserved.
0010b = fc2	00000b = **fc2.0**. All other encodings are reserved.
0011b = fc3	00000b = **fc3.0**. 00001b = **fc3.1**. 00010b = **fc3.2**. 00011b = **fc3.3**. All other encodings are reserved.
0100b = fc4	00000b = **fc4.0**. All other encodings are reserved.

These are special hardware registers used in handling flow control operations. These registers may be accessed ONLY in context save/restore operation using the SIP. These registers are accessed with the 'MOV' opcode. Use of any other opcode or access of these registers in non-context save/restore modes may result in undeterministic behaviour of hardware.

These registers are accessed as 256b registers. Parts of the 256b register may be redundant, depending on the hardware implementation of each register. The fields "RegNum" and "SubRegNum" are used together to address these registers.

Immediate

Two forms of immediate are provided as a source operand: scalar and vector.

Description
The immediate field may be 64 bits or 32 bits. For a word, unsigned word, or half-float immediate data, software must replicate the same 16-bit immediate value to both the lower word and the high word of the 32-bit immediate field in a GEN instruction. The 64-bit immediate takes up two DWords of the instruction bit field. Hence a 64-bit immediate is supported ONLY for a MOV operation. The field is denoted by ***imm32:type*** for 32-bit immediates and ***imm64:type*** for 64-bit immediates.

Description
For a scalar immediate, the numeric data types supported are ***:uw, :w, :ud, :d, :uq, :q*** for integers AND ***:hf, :f, :df*** for floats. Refer to the Instruction Machine format topics for the encoding of these immediates.

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. Both integer and float immediate vectors are supported.

An immediate integer vector is denoted by type **v** or **uv** as *imm32:v* or *imm32:uv*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Refer to the Numeric DataType topic for description of the packing of vector integers to a DWord.

An immediate float vector is denoted by type **vf** as *imm32:vf*, where the 32-bit immediate field is partitioned into 4 8-bit subfields. Refer to the Numeric DataType topic for the description of the packing of vector floats to a DWord.

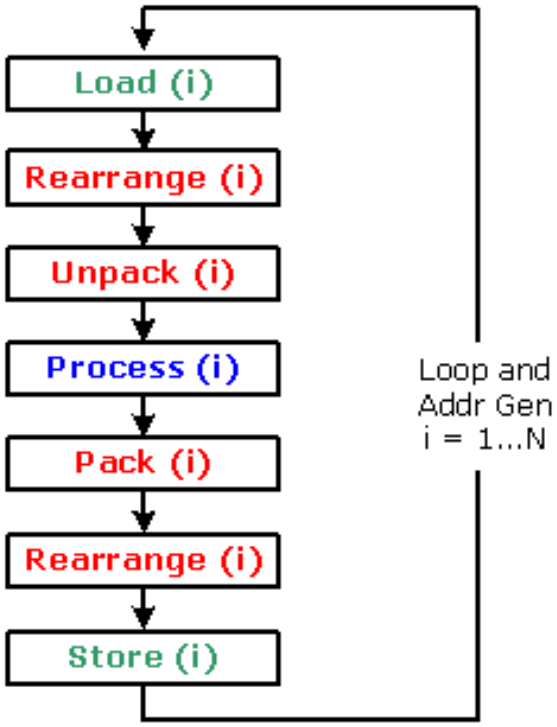
When an immediate vector is used in an instruction, the destination must be 128-bit aligned with destination horizontal stride equivalent to a word for an immediate integer vector (***v***) and equivalent to a DWord for an immediate float vector (***vf***).

Region Parameters

Unlike conventional SIMD architectures where an N-bit wide SIMD instruction can only operate on N-bit aligned SIMD data registers, a region-based register addressing scheme is employed in GEN architecture. The region-based register addressing capability significantly improves the SIMD computation efficiency by providing per-instruction-based multiple data gathering from register file. This avoids instruction overhead to perform data pack, unpack, and shuffling, which has been observed on other SIMD architectures. One benefit of such capability is allowing various kinds of 3D Graphics API Shader compute models to run efficiently on GEN. Another benefit is allowing high throughput of media applications, which tend to operate on byte or word data elements.

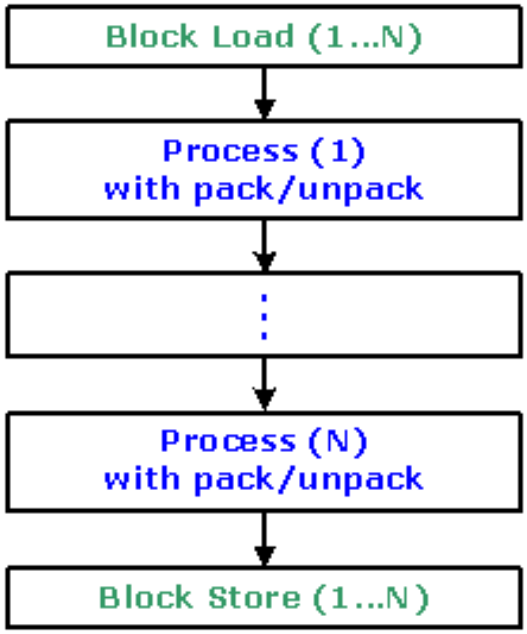
This can be illustrated by the example shown in *Region Parameters* and *Region Parameters*. As shown in *Region Parameters*, a sequence of SIMD instruction is executed on a conventional load/store based superscalar machine with SIMD instruction extension. The data parallelism can be achieved by first level of loop unrolling. As shown, there is a second level of loop for the task. Before a given SIMD compute instruction, *Process (i)*, can proceed, there might be a load, a data rearrange and a data unpack (and conversion) instruction to load and prepare the input data. After the compute instruction is complete, it might also require pack, re-arrange and store instructions, to format and save the same to memory. At the loop, other scalar computations such as loop count and address generation may be needed. For the same program, when the data can fit in the large GEN GRF register file, the outer loop may be unrolled for GEN. Here one or a few block loads (using *send* instruction) may be sufficient to move the working set into GRF. Then the data shuffle can be combined with the processing operation with region-based addressing capability. Per operand float type and mixed data type operation may also allow GEN to combine data conditioning operations with computing operations. These techniques in GEN architecture help to achieve high compute efficiency and throughput for graphics and media applications.

Conventional SIMD Instruction Sequence



B6899-01

GEN SIMD Instruction Sequence for the Same Program



B6900-01

In a GEN instruction, each operand defines a region in the register file. A region may contain multiple data elements. Each data element is assigned to an execution channel in the EU. The total number of data elements of a region is called the **size** of the region, or the size of the operand. The number of execution channels is called the **execution size** (*ExecSize*), which is specified in the instruction word. ExecSize determines the size of region for source and destination operands in an instruction.

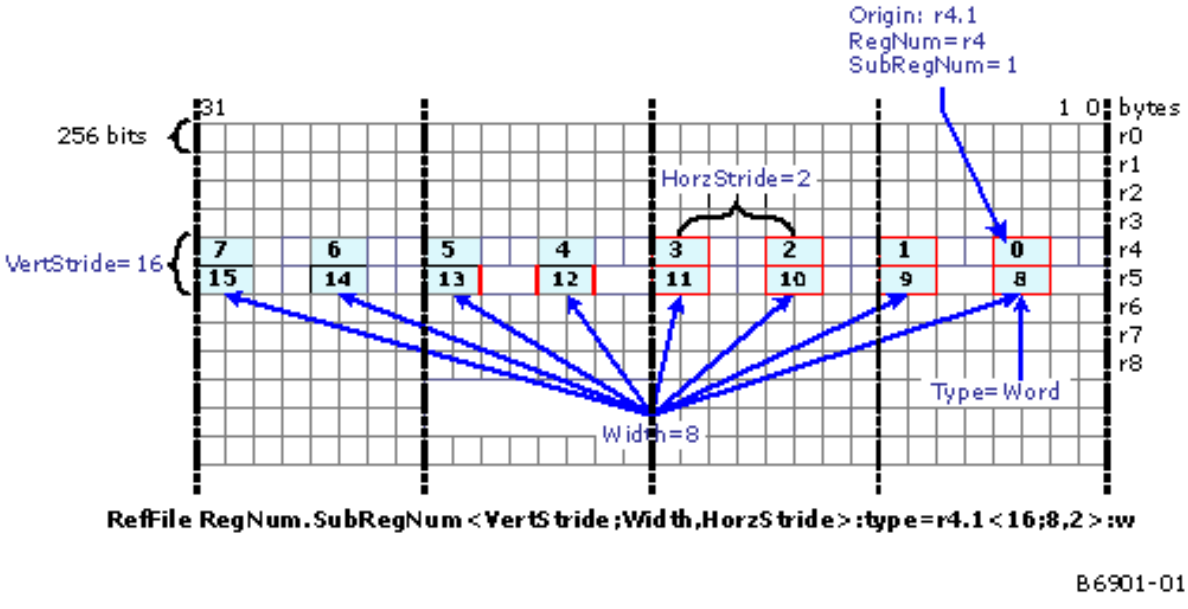
- For an instruction with two source operands, the sizes of the two source operands must be the same.
- The size of a destination operand generally matches the execution size, therefore equals to the number of source operand(s) in the same instruction.
    - Exception of this rule is present for the integer reduction instructions (such as sad2 and sada2) where the destination area is smaller than the source area.

Regions are **generalized 2-dimensional** (2D) arrays in row-major order. The first dimension is named the **horizontal** dimension (data elements within a row) and the second dimension is termed the **vertical** dimension (data elements in a column). Here, horizontal/vertical and row/column are just symbolic notations.

Description
When the GRF registers are viewed as a row-major 2D array of memory, such a notation normally matches well with the geometric locations of the data elements of an operand.

However, as the register region is fully described by the parameters discussed below, the data elements of a register region may not form a regular rectangular shape. For example, Vertical Stride parameter is allowed to be smaller than Horizontal Stride, making the rows of a register region interleave with each other. It should also note that the meanings of horizontal/vertical here is different than that used for the flag control in Section Flag Register

An example of a register region (*r4.1<16;8,2>:w*) with 16 elements

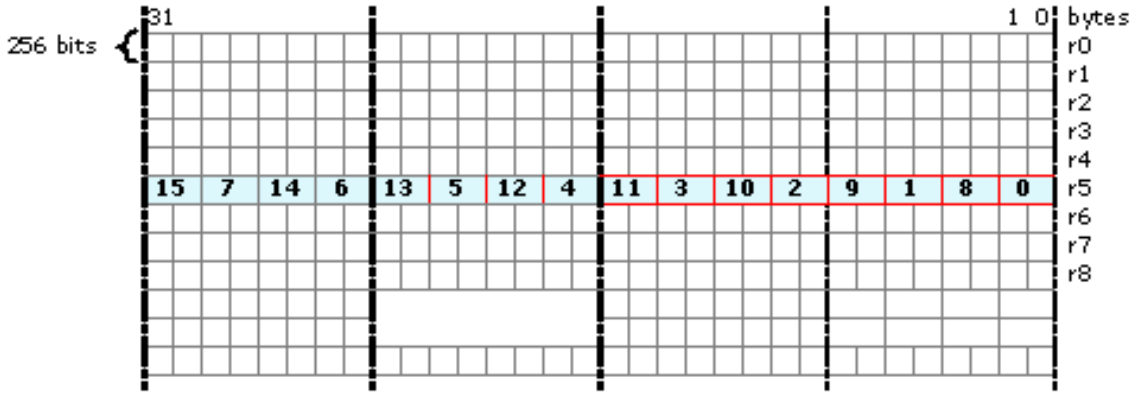


B6901-01

Region Parameters shows another example where the rows are interleaved. The region, having word data elements, starts at location r5.0:w. HorzStride, the distance within a row, is 2 words. So the second element (channel number 1) is at location 5.2:w. And there are 8 elements per row. VertStride, the distance between two rows, is only 1 word, which is less than HorzStride. Therefore, the first element of the second row (channel number 8) is at r5.1:w, just next to channel number 0. It is clear from the picture that the two rows are interleaved.

By varying the region parameters, reader may construct other configurations. The next section provides more details on the region-based register addressing. However, there are restrictions imposed by hardware implementation, which can be found in the later sections of this chapter.

A 16-element register region with interleaved rows (*r5.0<1;8,2>:w*)



Without considering the source channel swizzle and destination register region description, the above row-major-order region description provides the data assignment to each execution channel. The following pseudo code computes the addresses of data elements assigned to execution channels for a special case when the destination register is aligned to 256-bit register boundary.

```
// Input: Type: ub | b | uw | w | ud | d | f | v
//RegNum: In unit of 256-bit register
//SubRegNum: In unit of data element size
//ExecSize, Width, VertStride, HorzStride: In unit of data elements
// Output: Address[0:ExecSize-1] for execution channels
int ElementSize = (Type=="b"||Type=="ub") ? 1 : (Type=="w"|Type=="uw") ? 2 : 4;
int Height = ExecSize / Width;
int Channel = 0;
int RowBase = RegNum«5 + SubRegNum * ElementSize;
for (int y=0; y<Height; y++) {
```

```
        int Offset = RowBase;
        for (int x=0; x<Width; x++) {
Address [Channel++] = Offset;
Offset += HorzStride*ElementSize;
        }
        RowBase += VertStride * ElementSize;
}
```

As *HorzStride* and *VertStride* are specified independently (note that *VertStride* might be smaller than or equal to *HorzStride*), the region may take various shapes from a replicated scalar, a replicated vector, a vector of replicated scalars, a sliding window, to a non-overlapped 2D array.

A region-based description of a destination operand can take the following simplified format

RegFile RegNum.SubRegNum<HorzStride>:type

The destination operand is only allowed to have a 1 dimensional region. The Register Region Origin and Type are the same as for a source operand. The total number of elements is given by *ExecSize*. However, only *HorzStride* is required to describe the 1D region, not *VertStride* and *Width*.

As a source register region may cross multiple physical GRF registers, an instruction with such source operands may take more than two execution cycles to gather source data elements for execution. The destination register region is restricted to be within a physical GRF register. In other words, destination scatter writes over multiple registers are not supported.

Region Addressing Modes

There are two different register addressing modes: Direct register addressing and register-indirect register addressing. Depending on the register region description, the register-indirect register addressing mode can be further divided into three usages: 1x1 index region where only the origin of register region is provided by the address register, Vx1 index region where the offset of each row of the register region is provided by an address register, VxH index region where the offset of each data element is provided by an address register.

Direct Register Addressing

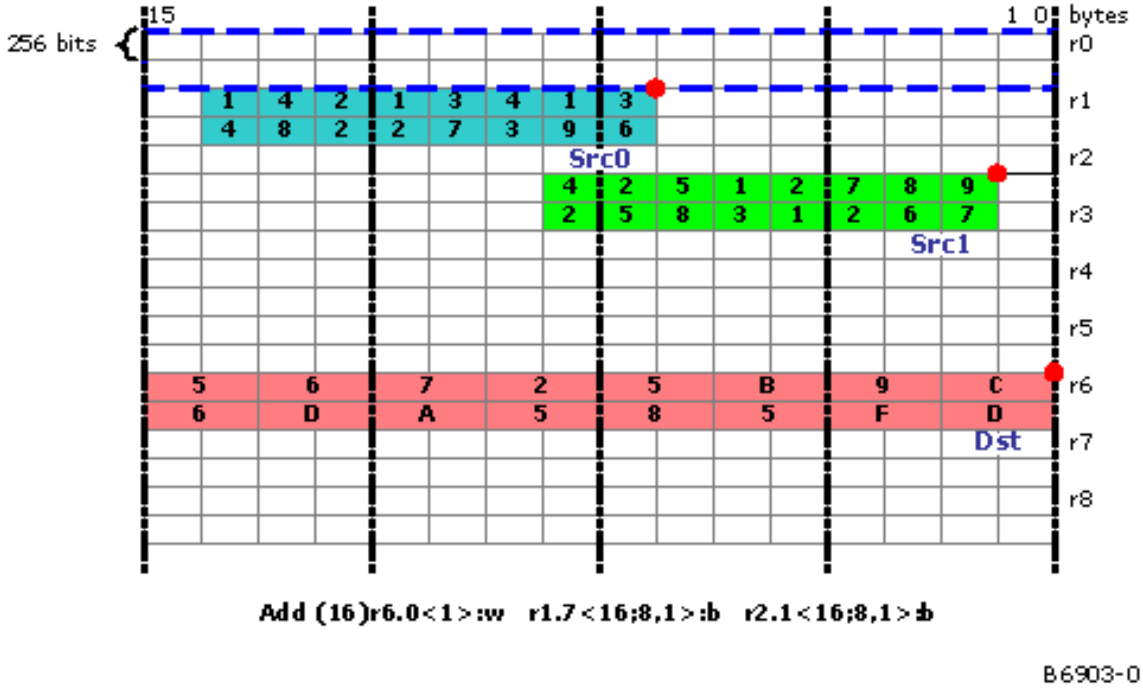
In this mode, all register region parameters are specified for an operand using fields in the instruction word.

Direct Register Addressing and *Direct Register Addressing* are two examples of direct register addressing.

For the example in *Direct Register Addressing*, all operands are 2D rectangular regions having the same size of 16 data elements. The two source operands, *Src0* and *Src1*, have 16 bytes. The destination operand, *Dst*, has 16 words. There are 8 elements in a row for *Src0* and *Src1*. The vertical stride of 16 bytes for *Src0* and *Src1* indicates that the first element and the 9'th element are 16 bytes apart in the register file. Note that *Src0* falls into the 256-bit physical GRF register starting at r1.0, but Src1 crosses the

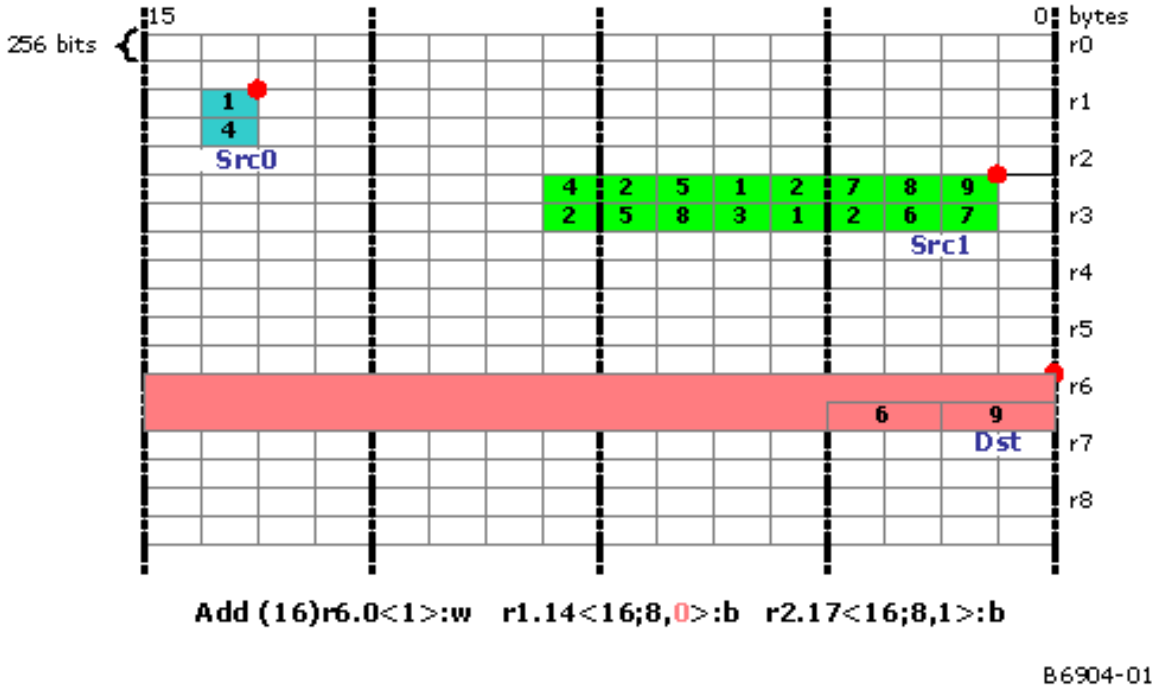
256-bit physical GRF register boundary between r2 and r3. The numbers in the shaded regions are the values of the data elements. Observing the upper right corners of the source/destination regions (first data element), we have C = 3+9.

A region description example in direct register addressing



For the example in *Direct Register Addressing*, the sizes of areas of *Src0* and *Src1* are the same, but *Src0* contains a vector of replicated scalars. With HorzStride = 0 and Width = 8, the first row of 8 elements in Src0 is a replication of the byte at r1.14. Comparing *ExecSize* of 16 to Width of 8 indicates that there is a second row of 8 elements in *Src0*. With VertStride = 16, the second row in *Src0* is a replication of the byte at r1.20 (20 = 14+16). Effectively, the 16 data elements of *Src0* are {1,1,1,1,1,1,1,1, 4,4,4,4,4,4,4,4}.

A region description example in direct register addressing with src0 as a vector of replicated scalars

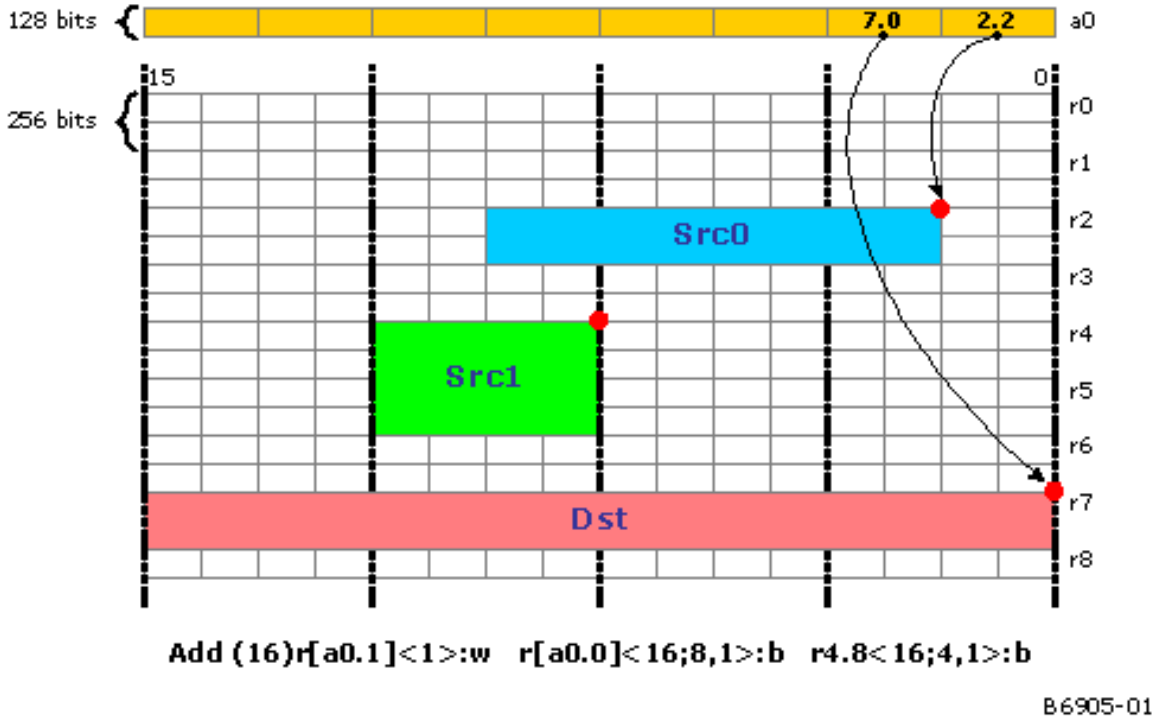


Register-Indirect Register Addressing with a 1x1 Index Region

In the register-indirect register addressing mode with 1x1 index region, the region origin is provided by the content of the address register, the rest of region parameters are provided by the fields in the instruction word.

Register-Indirect Register Addressing with a 1x1 Index Region depicts an example for this addressing mode. For example, the presence of a full region description <16;8,1> for Src0 indicates that only the origin of the region is provided by the address register a0.0.

An example illustrating register-indirect register addressing mode with a 1x1 index region



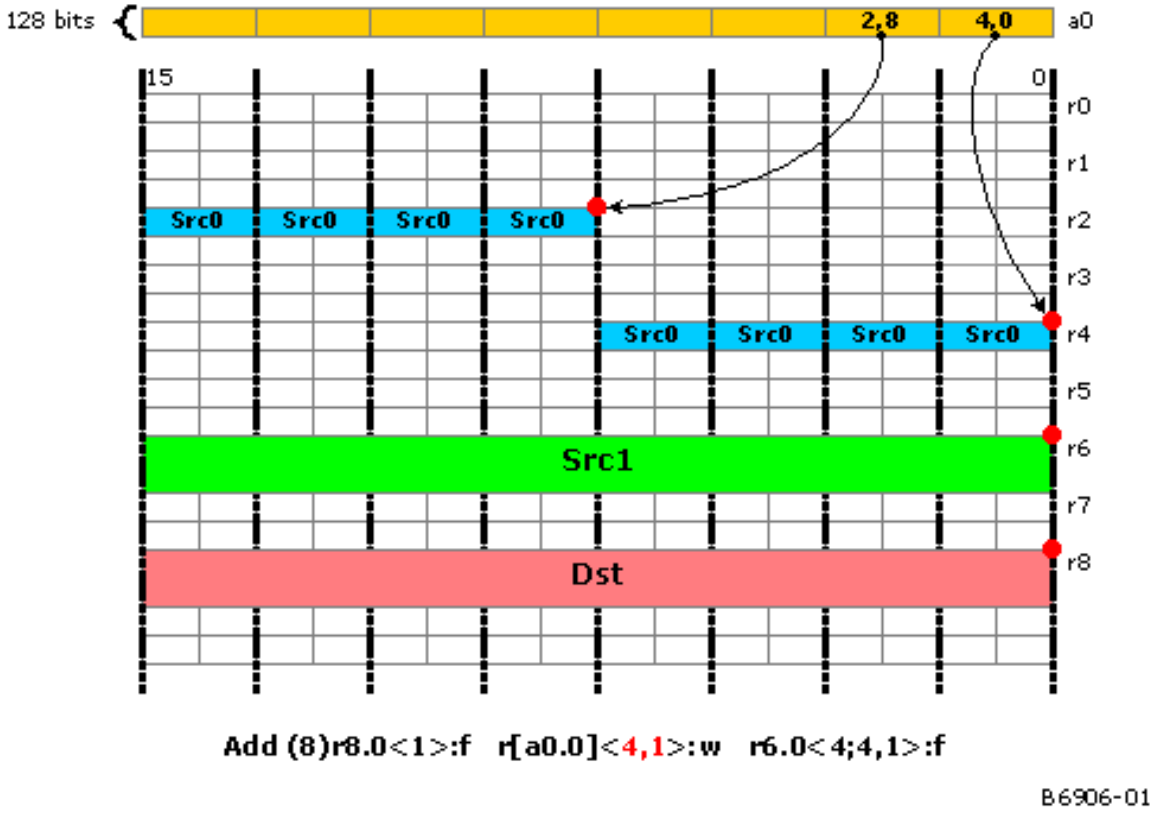
Register-Indirect Register Addressing with a Vx1 Index Region

In the register-indirect register addressing mode with Vx1 index region, the horizontal dimension is described by the fields in the instruction word and the vertical dimension is described by an address register region. Specifically, the origin of each row of the data region is provided by the contents of an address register region. The rows are described by the width and the horizontal stride. The first address register is provided and the following contiguous address registers are for the following rows. The total number of address registers used is inferred from the parameters *ExecSize* and *Width*.

Within the 16-bit address register, bits 15:5 determine RegNum and bits 4:0 determine SubRegNum.

An example is provided in *Register-Indirect Register Addressing with a Vx1 Index Region*. The assembly syntax notion of a register region without vertical stride, <4,1>, corresponding to the special encoding of vertical stride of 0xF in the instruction word, indicates the VxH or Vx1 mode of indirect register addressing. In this case, the origin for each row of src0 is provided by the address register. As ExecSize/Width = 2, there are two address registers a0.0 and a0.1, each pointing to a row of 4 data elements.

An example illustrating register-indirect-register addressing mode with a Vx1 index region (src0)

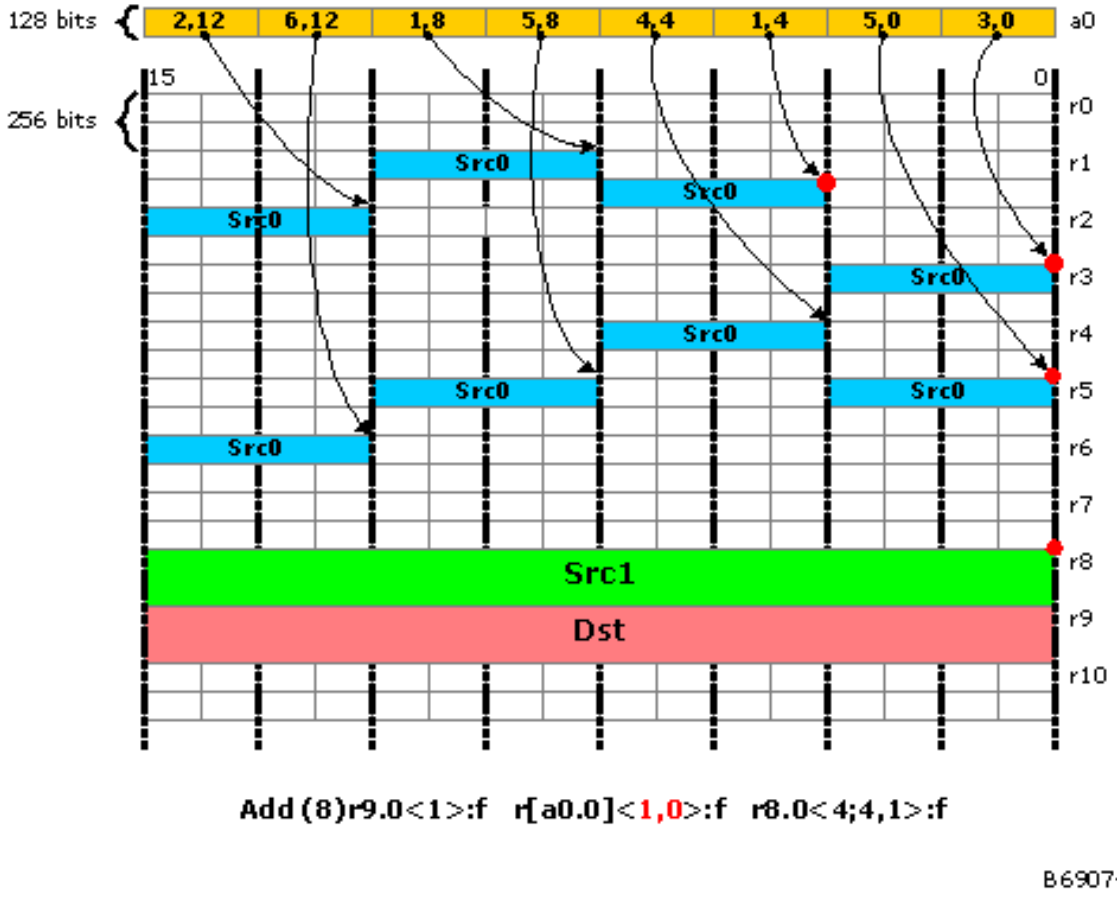


Register-Indirect Register Addressing with a VxH Index Region

In the register-indirect register addressing mode with VxH index region, the position of each data element is provided by the contexts in an address register region. This mode has the identical syntax as the Vx1 index region mode, and in fact, can be viewed as a special case of the Vx1 mode. When *Width* of the region is 1, the number of address registers used equals *ExecSize*.

An example is provided in *Register-Indirect Register Addressing with a VxH Index Region*. The absent of vertical stride in the region description <1,0> with width = 1 indicates that the origin for each row of 1 data element of Src0 is provided by the address register. As ExecSize/Width = 8, there are 8 address registers from a0.0 to a0.7, each pointing to a single data elements.

An example illustrating register-indirect register addressing mode with a VxH index region (Src0).



Access Modes

There are two basic GEN register access modes controlled by a single bit instruction subfield called Access Mode.

- 16-byte Aligned Access Mode (**align16**): In this mode, the origins of all operands (sources and destination), whether it is by direct addressing or register-indirect addressing, are 16-byte aligned. For example a row in the region description starts at 16-bype aligned and the width the row must be 4 and the 4 data elements within a row must span 16-bytes. In this access mode (and with other restrictions put forward later), full-channel swizzle for both source operands and per-channel mask for destination operand are supported on a 4-component basis. In other words, the control and setting of full source swizzle and destination mask are repeated for every 4 components up to total of *ExecSize* channels.
    - The **align16** access mode can be used for AOS operations. See examples provided in the Primary Usage Model section for SIMD4x2 and SIMD4x1 modes of operation to support 3D API Vertex Shader and Geometric Shader execution.
- 1-byte Aligned Access Mode (**align1**): In this mode, the origins of all operands may be aligned to their data type and could be 1-byte if the operand is of byte type. In this access mode, full region register descriptions are supported, however, source swizzle or destination mask are not supported.
    - The **align1** access mode can be used for SOA operations. See examples provided in the Primary Usage Model section for SIMD8 and SIMD16 modes of operation to support 3D API Pixel Shader. Many media applications also operate well in **align1** access mode.
-

Execution Data Type

The GEN architecture carries out arithmetic and logical operations using a smaller set of data types than the variety supported as source or destination operands. These are the *execution data types*. A particular arithmetic or logical instruction has one execution data type, from those listed in the table.

Execution Data Types

Type	Description
W	Word. 16-bit signed integer.
D	Doubleword. 32-bit signed integer.
Q	Quadword. 64-bit signed integer.
F	Float. 32-bit single precision floating-point number.
DF	Double Float. 64-bit double precision floating-point number.
HF	Half Float. 16-bit half precision floating-point number.

The following rules explain the conversion of multiple source operand types, possibly a mix of different types, to one common execution type:

- For floating-point sources, all source operands must have the same floating-point type, with the exceptions below:
  - A two-source floating-point instruction can have Float as the src0 type and VF (Packed Restricted Float Vector) as the immediate src1 type.
- Mixing floating-point and integer source types is not allowed. Either all source types must be one floating-point type or all source types must be integer types.
- Unsigned integers are converted to signed integers.
- Byte (B) or Unsigned Byte (UB) values are converted to a Word or wider integer execution type.
- If source operands have different integer widths, use the widest width specified to choose the signed integer execution type.

Note that when the execution data type is an integer type, it is always a signed integer type. For integer execution types, extra precision is provided within the hardware, including the accumulators, so that conversions from unsigned to signed do not affect instruction correctness.

Register Region Restrictions

A register region is described as *packed* if its elements are adjacent in memory, with no intervening space, no overlap, and no replicated values. If there is more than one element in a row, elements must be adjacent. If there is more than one row, rows must be adjacent. When two registers are used, the registers must be adjacent and both must exist.

The following register region rules apply to the GEN implementation.

1. **General Restrictions Based on Operand Types**

   There are these general restrictions based on operand types:

   1. Where *n* is the largest element size in bytes for any source or destination operand type, *ExecSize* * *n* must be <= 64.
   2. When the Execution Data Type is wider than the destination data type, the destination must be aligned as required by the wider execution data type and specify a *HorzStride* equal to the ratio in sizes of the two data types. For example, a *mov* with a D source and B destination must use a 4-byte aligned destination and a *Dst.HorzStride* of 4.

2. **General Restrictions on Regioning Parameters**

   The mapping of data elements within the region of a source operand is in row-major order and is determined by the region description of the source operand, the destination operand, and the *ExecSize*, with these restrictions:

   1. *ExecSize* must be greater than or equal to *Width*.
   2. If *ExecSize* = *Width* and *HorzStride* ≠ 0, *VertStride* must be set to *Width* * *HorzStride*.
   3. If *ExecSize* = *Width* and *HorzStride* = 0, there is no restriction on *VertStride*.
   4. If *Width* = 1, *HorzStride* must be 0 regardless of the values of *ExecSize* and *VertStride*.
   5. If *ExecSize* = *Width* = 1, both *VertStride* and *HorzStride* must be 0.
   6. If *VertStride* = *HorzStride* = 0, *Width* must be 1 regardless of the value of *ExecSize*.
   7. *Dst.HorzStride* must not be 0.
   8. *VertStride* must be used to cross GRF register boundaries. This rule implies that elements within a '*Width*' cannot cross GRF boundaries.

3. **Region Alignment Rules for Direct Register Addressing**
   1. In Direct Addressing mode, a source cannot span more than 2 adjacent GRF registers.
   2. A destination cannot span more than 2 adjacent GRF registers.
   3. When a source or destination spans two registers, there are restrictions that vary by project, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

      ### Source or Destination Spans Two Registers

4. **Special Cases for Byte Operations**

1. When the destination type is byte (UB or B) only a 'raw move' using the *mov* instruction supports a packed byte destination register region: *Dst.HorzStride* = 1 and *Dst.DstType* = (UB or B). This packed byte destination register region is not allowed for any other instructions, including a 'raw move' using the *sel*instruction, because the *sel* instruction is based on Word or DWord wide execution channels.
2. There is a relaxed alignment rule for byte destinations. When the destination type is byte (UB or B), destination data types can be aligned to either the lowest byte or the second lowest byte of the execution channel. For example, if one of the source operands is in word mode (a signed or unsigned word integer), the execution data type will be signed word integer. In this case the destination data bytes can be either all in the even byte locations or all in the odd byte locations.

   This rule has two implications illustrated by this example:

```
// Example:
mov (8) r10.0<2>:b r11.0<8;8,1>:w
mov (8) r10.1<2>:b r11.0<8;8,1>:w

// Dst.HorzStride must be 2 in the above example so that the destination
// subregisters are aligned to the execution data type, which is :w.
// However, the offset may be .0 or .1.
// This special handling applies to byte destinations ONLY.
```

5. **Special Cases for Word Operations**

There are some special cases for word operations for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may not show and there are no special cases in this category.

There is a relaxed alignment rule for word destinations. When the destination type is word (UW, W, HF), destination data types can be aligned to either the lowest word or the second lowest word of the execution channel. This means the destination data words can be either all in the even word locations or all in the odd word locations.

```
// Example:
add (8)  r10.0<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf
add (8)  r10.1<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf

// Note: The destination offset may be .0 or .1 although the destination
subregister
// is required to be aligned to execution datatype.
```

6. **Special Requirements for Handling Double Precision Data Types**

There are special requirements for handling double precision data types that vary by project, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Special Requirements for Handling Double Precision Data Types

Requirement

In Align16 mode, all regioning parameters must use the syntax of a pair of packed floats, including channel selects and channel enables.
`// Example:`
`mov (8) r10.0.xyzw:df r11.0.xyzw:df`
`// The above instruction moves four double floats. The .x picks the`
`// low 32 bits and the .y picks the high 32 bits of the double float.`

In Align1 mode, all regioning parameters like stride, execution size, and width are in units of element size. However in Align16 mode, the channel selects and channel enables must always be used in pairs of packed floats, because these parameters are defined for DWord elements ONLY.
`// Example:`
`mov (4) r10.0<1>:df r11.0<4;4,1>:df`
`// The above instruction moves four double floats.`
`// Note the difference between [DevIVB] and [DevHSW+].`

7. **Special Requirements for Handling Mixed Mode Float Operations**

   There are some special requirements for handling mixed mode float operations for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Requirement
In Align16 mode, when half float and float data types are mixed between source operands OR between source and destination operands, the register content are assumed to be packed. In such cases the execution size reflects the number of float elements. Since a stride of 1 is assumed, source is selected in packed form and 16 bit packed data is updated on the destination operand, if the datatype is half-float.

For Align16 mixed mode, both input and output packed f16 data must be oword aligned, no oword crossing in packed f16.
Examples:
`Case (a)`
`mad (8) r10.0.xy:hf r11.0.xxxx:f r12.xyzw:hf r13.yyyy:hf`
`// The 16b of each word (r12.0, r12.1, r12.2, r12.3.. and so on) forms the source operand.`
`// r13.1 and r13.5 is replicated for source operand.`
`// The lower 16b of a Dword is updated for destination. With channel enables .xy , r10.0, r10.1, r10.4 and r10.5 are updated.`
 	
`Case (b)`
`mad (8) r10.0.xy:f r11.0.xxxx:f r12.xyzw:hf r13.yyyy:hf`
`// The example is similar to Case(a), except that entire DWord is updated on the destination.`

In Align16 mode, replicate is supported and is coissueable. `mad(8) r20.xyzw:hf r3.0.r:f r6.0.xyzw:hf r6.0.xyzw:hf {Q1}`

Requirement

No SIMD16 in mixed mode when destination is packed f16 for both Align1 and Align16.
`mad(8) r3.xyzw:hf r4.xyzw:f r6.xyzw:hf r7.xyzw:hf`
`add(8) r20.0<1>:hf r3<8;8,1>:f r6.0<8;8,1>:hf {Q1}`

No accumulator read access for Align16 mixed float.
When source is float or half float from accumulator register and destination is half float with a stride of 1, the source must register aligned. i.e., source must have offset zero.

No swizzle is allowed when an accumulator is used as an implicit source or an explicit source in an instruction. i.e. when destination is half float with an implicit accumulator source, destination stride needs to be 2.
`mac(8) r3<2>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`
`mov(8) r3<1>:f acc0.0<8;4,2>:hf`

In Align16, vertical stride can never be zero for f16 `add(8) r3.xyzw:hf r4.0<4>xyzw:f r6.0<0>.xyzw:hf`

Math operations for mixed mode:
- In Align16, only packed format is supported
`math(8) r3.xyzw:hf r4.0.<4>xyzw:f r6.0<0>.xyzw:hf 0x09`
- In Align1, f16 inputs need to be strided
`math(8) r3<1>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`

In Align1, destination stride can be smaller than execution type. When destination is stride of 1, 16 bit packed data is updated on the destination. However, output packed f16 data must be oword aligned, no oword crossing in packed f16. `add(8) r3<1>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`
For mixed float operations, f16 datatype write to accumulator cannot be packed destination.

8. **Regioning Rules for Register Indirect Addressing**

   Regioning rules for register indirect addressing vary for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Rules

1. When the execution size and destination regioning parameters require two adjacent registers, these registers are accessed using one index register ONLY.
`// Example:`
`mov (16) r[a0.0]:f r10:f`
`// The above instruction behaves the same as the following two instructions:`
`mov (8) r[a0.0]:f r10:f`

Rules

mov (8) r[a0.0, 8*4]:f r11:f
 	
2. When the destination requires two registers and the sources are 1x1 indirect mode, the sources must be assembled from two GRF registers accessed by a single index register. The data for each destination GRF register is entirely derived from one source register. This is ensured by appropriate use of regioning parameters. The exception to this is the use of indirect scalar sources, where the same element is used across the execution size.
// Example:
// Case (a)
add (16) r[a0.0]:f r[a0.2]:f r[a0.4]:f
// The above instruction behaves the same as the following two instructions:
add (8) r[a0.0]:f r[a0.2]:f r[a0.4]:f
add (8) r[a0.0, 8*4]:f r[a0.2, 8*4]:f r[a0.4, 8*4]:f
// Note that the immediate for the second instruction is based on regioning.
// In this case, it is 8 DWs.
 	
// Case (b)
add (16) r[a0.0]:ud r[a0.2]<4;8,1>:w r10<8;8,1>:ud
// The above instruction behaves the same as the following two instructions:
add (8) r[a0.0]:f r[a0.2]<4;8,1>:w r10<8;8,1>:ud
add (8) r[a0.0, 8*4]:f r[a0.2, 4*2]<4;8,1>:w r11<8;8,1>:ud
// Note that the immediate for the second instruction is based on regioning.
// VertStride of 4 with data type of word.
 	
// Case (c):
add (16) r[a0.0]:f r[a0.2]:f r[a0.4]<0;1,0>:f
// The above instruction behaves the same as the following two instructions:
add (8) r[a0.0]:f r[a0.2]:f r[a0.4]<0;1,0>:f
add (8) r[a0.0, 8*4]:f r[a0.2, 8*4]:f r[a0.4]<0;1,0>:f
// Note that the src1 indirect address does not change.
 	
3. Indirect addressing on src1 must be a 1x1 indexed region mode.
4. When a Vx1 or a VxH addressing mode is used on src0, the destination may use one or two registers.
// Example:
// Case (a)
add (16) r[a0.0]<1>:d r[a0.0]<4,1>:ud r16.0<8;8,1>:ud
// The above instruction behaves the same as the following two instructions:
add (8) r[a0.0]<1>:d r[a0.0]<4,1>:ud r16.0<8;8,1>:ud
add (8) r[a0.0, 8*4]<1>:d r[a0.2]<4,1>:ud r17.0<8;8,1>:ud
// Since the pointer (index register) is incremented every 4 elements
// (width), the second instruction moves from a0.0 to a0.2.
 	
// Case (b)
add (16) r10.0<2>:uw r[a0.0, 0]<1,0>:uw r16.0<8;8,1>:uw
// The above instruction behaves the same as the following two instructions:
add (8) r10.0<2>:uw r[a0.0, 0]<1,0>:uw r16.0<8;8,1>:uw
add (8) r11.0<2>:uw r[a0.8, 0]<1,0>:uw r17.0<8;8,1>:uw
// Since the pointer (index register) is incremented every 1 element
// (width), the second instruction moves from a0.0 to a0.8.
 	
5. Indirect addressing on the destination must be a 1x1 indexed region mode.

Execution size of 32 is NOT supported in Vx1 or VxH modes.

1. **Special Restrictions**

There are some special restrictions on register region access for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Restriction
All flow control (branching) instructions must use the Align1 access mode.
When using Align16 mode for conversion of data elements of different sizes, both source and destination must be one register each.

In Align16 mode, each destination register gets all data from one source register. This means, the data for one destination register is never scattered across two source registers.
`// Example:`
`// Allowed – all sources are contained within one register.`
`mul (8)  r10.0:f  r11.0:f  r12.4<0>:f`
 	
`// NOT Allowed – src1 (r14) is scattered across two registers.`
`mad (8) r10.0:f r12.0<0>:f r14.4:f r16.0:f`

Conversion between Integer and HF (Half Float) must be DWord-aligned and strided by a DWord on the destination.
`// Example:`
`add (8) r10.0<2>:hf r11.0<8;8,1>:w r12.0<8;8,1>:w`
`// Destination stride must be 2.`
`mov (8) r10.0<2>:w r11.0<8;8,1>:hf`
`// Destination stride must be 2.`

The src, dst overlapping behavior with the second half src and the first half destination to the same register must not be used with any compressed instruction.

Destination Operand Description

Destination Region Parameters

Based on the above restrictions, a subset of register region parameters are sufficient to describe the destination operand:

- Destination Register Origin
  - Destination Register Number and Destination Subregister Number for direct register addressing mode
  - A Scalar Destination Register Index for register-indirect-register addressing mode
- Destination Register ‘Region’ – Note that destination register region does not have full region description parameters
  - Destination Horizontal Stride

SIMD Execution Control

This section of the BSpec discusses SIMD execution, both with and without predication. See the subtopics for more details.

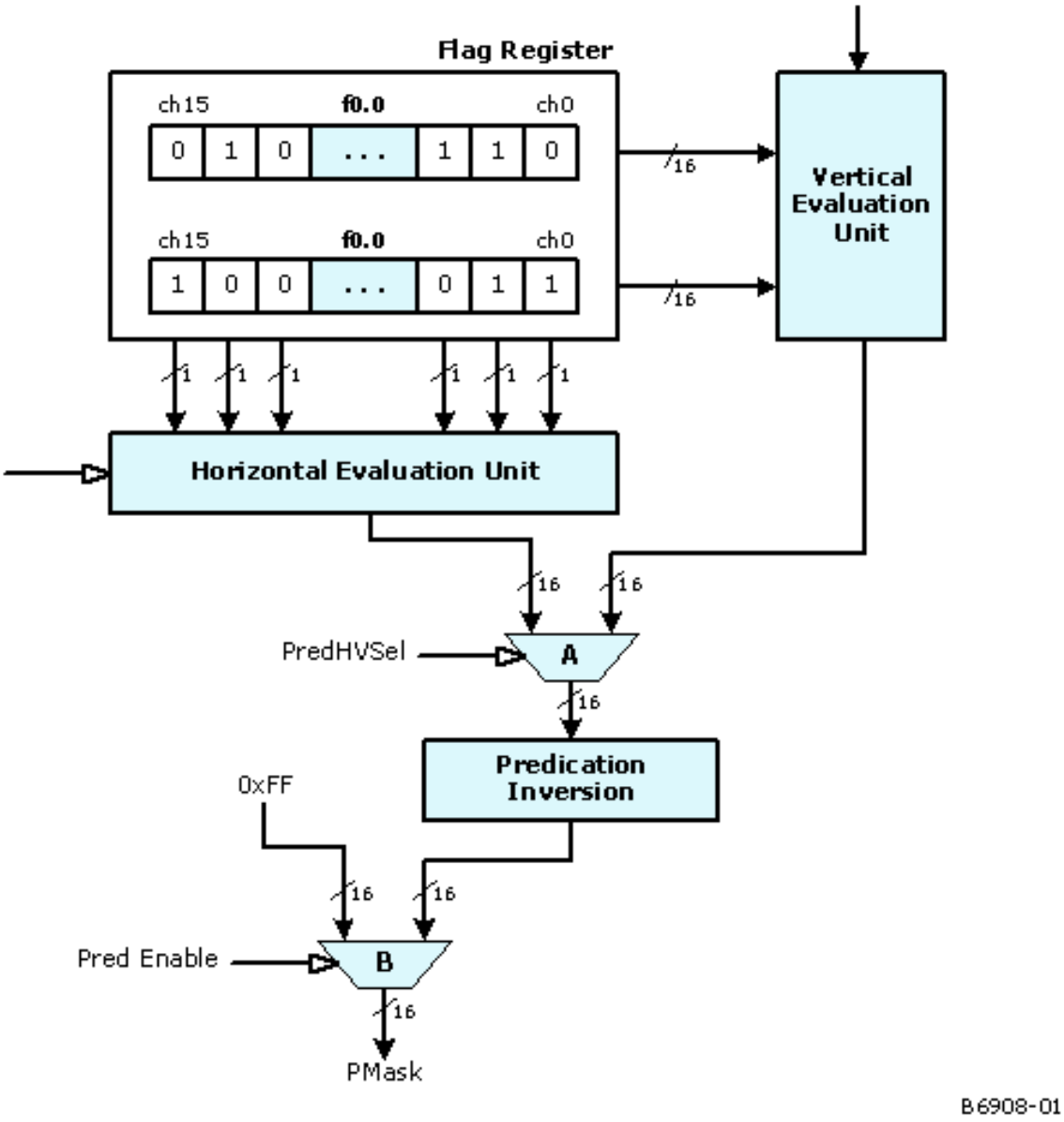
Predication

Predication is the conditional SIMD channel selection for execution on a per instruction basis. It is an efficient way of dynamic SIMD channel enabling without paying branch instruction overhead. When predication is enabled for an instruction, a Predicate Mask (PMask), which contains 16-bit channel enables, is generated internally in EU. Note that PMask is not a software visible register. It is provided here to explain how SIMD execution control works. PMask generation is based on the Predication Control (*PredCtrl*) field, Predication Inversion (*PredInv*) field and the flag source register in the instruction word. See Instruction Summary chapter for definition of these fields.

*Predication*shows the block diagram of the hardware logic to generate PMask. PMask is generated based on combinatory logic operation of the bits in the flag register. Instruction field *PredCtrl* controls the horizontal evaluation unit and vertical evaluation unit. MUX A in the figure selects whether horizontally-evaluated results or vertically-evaluated results are sent to the Predication Invertion unit. The *PredInv* field controls the Prediction Inversion unit. Either one 16-bit flag subregister or the whole flag register may be selected to generate the PMask depending on the predication control modes. MUX B indicates that predication can be enabled and disabled. Predication can be grouped into the following three categories. Predication functionality also depends on the Access Mode of the instruction.

- No predication: Of course, predication can be disabled. This is the most commonly used case.
- Predication with horizontal combination: the predicate mask is generated based on combinatory logic operation of bits within a selected flag subregister.
- Predication with vertical combination: the predicate mask is generated based on combinatory logic operation of bits across flag multiple subregisters.

Generation of predication mask



No Predication

When PredCtrl field of a given instruction is set to 0 (“no predication”), it indicates that no predication is applied to this instruction. Effectively, the resulting PMask is all 1’s. This is shown by the 2:1 multiplexer B controlled by the Pred Enable signal in *Predication*. Where predication is not enabled for an instruction, multiplex B is selected to output 0xFF to PMask.

Predication with Horizontal Combination

Predication with horizontal combination inputs the 16 bits of a single flag subregister (f0.0:uw or f0.1:uw) and passes them through combinatory logic of the Horizontal Evaluation unit to create PMask.

The simplest combination is 'no combination' – the same 16 bits from selected flag subregister are output to MUX A. In this case, a bit in the selected flag subregister controls the conditional execution of the corresponding execution channel. Let the selected flag subregister be denoted as f0.#, the following pseudo code describes the predicate mask generation for predication with sequential flag channel mapping.

```
If (PredCtrl == “Sequential flag channel mapping”) {
    For (ch=0; ch<16; ch++)
        PMask[ch] = (PredInv == TRUE) ? ~f0.#[ch] : f0.#[ch];
}
```

More complex horizontal evaluation is based on channel grouping. A group of adjacent channels (bits from flag subregister) are evaluated together and a single bit is replicated to the group. The size of groups is in power of 2. The supported combination depends on the Access Mode of an instruction.

In **Align16** access mode, horizontal combination is based on 4-channel groups.

- Channel replication: PredCtrl of '.x', '.y', '.z' and '.w' select a single channel from each 4-channel group and replicate it as the output for the group. For example, PredCtrl = '.x' means that channel 0 in each group is replicated.
- OR combination: PredCtrl of '.any4h' means that if **any** of the channel in a group is enabled, outputs for the 4 channels in the group are all enabled.
- AND combination: PredCtrl of '.all4h' means that only when **all** of the channels in a group are enabled, the output for the group is enabled.

These combinations in **Align16** mode can be described by the following pseudo-code.

```
If (Access Mode == Align16) {
    For (ch = 0; ch < 16; ch += 4)
        Switch (PredCtrl) {
            Case     '.x':   bTmp = f0.#[ch]; break;
            Case     '.y':   bTmp = f0.#[ch+1]; break;
            Case     '.z':   bTmp = f0.#[ch+2]; break;
            Case     '.w':   bTmp = f0.#[ch+3]; break;
            Case '.any4h':   bTmp = f0.#[ch] | f0.#[ch+1] | f0.#[ch+2] | f0.#[ch+3]; break;
Case '.all4h':   bTmp = f0.#[ch] & f0.#[ch+1] & f0.#[ch+2] & f0.#[ch+3]; break;
        }
        bTmp = (PredInv == TRUE) ? ~bTmp : bTmp;
        PMask[ch] = PMask[ch+1] = PMask[ch+2] = PMask[ch+3] = bTmp;
    }
}
```

In **Align1** access mode, horizontal combination is based on AND combination '.any#h' and OR combination '.all#h' on channel groups with various sizes, where # is the number of channels in a group ranging from 2 to 16. This is described by the following pseudo-code.

```
If (Access Mode == Align1) {
    Switch (PredCtrl) {
        Case  '.any2h':   groupSize = 2; <op> = '|'; break;         Case  '.all2h':
 groupSize = 2; <op> = '&'; break;
        Case  '.any4h':   groupSize = 4; <op> = '|'; break;         Case  '.all4h':
 groupSize = 4; <op> = '&'; break;
        Case  '.any8h':   groupSize = 8; <op> = '|'; break;         Case  '.all8h':
 groupSize = 8; <op> = '&'; break;
        Case '.any16h':   groupSize = 16; <op> = '|'; break;         Case '.all16h':
 groupSize = 16; <op> = '&'; break;     }
    For (ch = 0; ch < 16; ch += groupSize) {
        For (inc = 0, bTmp = FALSE; inc < groupSize; inc ++)
            bTmp = bTmp <op> f0.#[ch+inc];
        For (inc = 0; inc < groupSize; inc ++)
            PMask[ch+inc] = bTmp;
    }
}
```

Predication with Vertical Combination

Predication with vertical combination uses both flag subregister as inputs. The AND or OR combination is across the subregisters on a channel by channel basis. This is shown by the following pseudo-code.

```
If (Access Mode == Align1) {
    For (ch = 0; ch < 16; ch ++) {
        If (PredCtrl == 'any2v')
            PMask[ch] = f0.0[ch] | f0.1[ch]
        Else If (PredCtrl == 'any2h')
            PMask[ch] = f0.0[ch] & f0.1[ch]
    }
}
```

Predication with Vertical Combination

Predication with vertical combination uses both flag register as inputs. The AND or OR combination is across the registers on a channel by channel basis. This is shown by the following pseudo-code.

```
If (Access Mode == Align1) {
    For (ch = 0; ch < 32; ch ++) {
        If (PredCtrl == 'anyv')
            PMask[ch] = f0.0[ch] | f1.0[ch]
        Else If (PredCtrl == 'allv')
            PMask[ch] = f0.0[ch] & f1.0[ch]
    }
}
```

End of Thread

There is no special instruction opcode (such as an END instruction) to cause the thread to terminate execution. Instead, the end of thread is signified by a *send* instruction with the end-of-thread (EOT) sideband bit set. Upon executing a *send* instruction with EOT set, the EU stops on the thread. Upon observing an EOT signal on the output message bus, the Thread Dispatcher makes the thread's resource available. If a thread uses pre-allocated resource managed by a fixed function, such as URB handles and

scratch memory, some fixed function protocol also requires the thread to terminate with the message header phase to carry the information in order for the fixed function to release the pre-allocated resource.

EU hardware guarantees that if a terminated thread has in-flight read messages or loads at the time of 'end' that their writebacks will not interfere with either other threads in the system or new threads loaded in the system in the future.

More details can be found in the *send* instruction description in Instruction Reference chapter.

Assigning Conditional Flags

Instructions can output two sets of conditional signals, one set from before the outputs clamping/re-normalizing/format conversion logic, we call this the pre conditional signals. The second set is generated from the final results after clamping and re-normalizing/format conversion logic, and we call this the post conditional signals. The post conditional signals are used for fusing the DirectX compare instruction. **Note:** The flags generated from the post conditional signals should be equivalent to the flags generated by a separate *cmp* instruction after the current arithmetic instruction.

The pre conditional signals are used to generated flags for *cmp*/*cmpn* instructions only, this logically does the compare of the two input sources. The post conditional signals are used to generated flags for all the other arithmetic instructions, this logically does the compare of the result with zero.

cmpn with both sources as NaNs is a don't care case as this doesn't impact the MIN/MAX operations.

The pre conditional signals include the following:

- **pre_sign** bit: This bit reflects the sign of the computed result before going through any kind of clamping, normalizing, or format conversion logic.
- **pre_zero** bit: This bit reflects whether the computed result is zero before any kind of clamping, normalizing, or format conversion logic.

The post conditional signals include the following:

- **post_sign** bit: This bit reflects the sign of the final result after all the clamping, normalizing, or format conversion logic.
- **post_zero** bit: This bit reflects whether the final result is zero after all the clamping, normalizing, or format conversion logic.
- **OF** bit: This bit reflects whether an overflow occured in any of the computation of the current instruction, including clamping, re-normalizing, and format conversion.
- **NC** bit: The NaN computed bit indicates whether the computed result is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0.
- **NS0** bit: The NaN Source 0 bit indicates whether src0 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0.

- **NS1** bit: The NaN Source 1 bit indicates whether src1 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0. For an operation with one source operand, this bit is also set to 0. This bit is only used for the comparison instruction *cmpn*, which is specifically provided to emulate MIN/MAX operations. For any other instructions, this bit is undefined.
- Note that the bits generated at the output of a compute are before the **.sat**.

Flag Generation for *cmp* Instructions (The Supported Conditional Modifiers are .e, .ne, .g, .ge, .l, and .le.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
.e	Equal-to	**(pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether the two sources are equal. If either source is NaN (i.e. NC is true), the flag is forced to false.
.ne	Not-Equal-to	**! (pre_zero & ! (NS0 | NS1)).** This conditional modifier test whether the two sources are equal. It takes exactly the reverse polarity as the modifier **.e**.
.g	Greater-than	**(! pre_sign & ! pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is greater than src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.ge	Greater-than-or-equal-to	**((! pre_sign | pre_zero) & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is greater than or equal to src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.l	Less-than	**(pre_sign & !pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is less than src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.le	Less-than-or-equal-to	**((pre_sign | pre_zero) & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is less than or equal to src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.

Flag Generation for All Instructions Other than *cmp/cmpn* Instructions (The Supported Conditional Modifiers are .e, .ne, .g, .ge, .l, .le, .o, and .u.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
.e	Equal-to	**(post_zero & ! NC).** This conditional modifier tests whether the result is equal to zero. If either source is NaN (i.e. NC is true), the flag is forced to false.
.ne	Not-Equal-to	**! (post_zero & ! NC).** This conditional modifier test whether the result is not equal to zero. It takes exactly the reverse polarity as modifier **.e**.
.g	Greater-than	**(! post_sign & ! post_zero & ! NC).** This conditional modifier tests whether result is greater than zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.ge	Greater-than-or-equal-to	**((! post_sign | post_zero) & ! NC).** This conditional modifier tests whether result is greater than or equal to zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.l	Less-than	**(post_sign & ! post_zero & ! NC).** This conditional modifier tests whether result is equal to zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.le	Less-than-or-equal-to	**((post_sign | post_zero) & ! NC).** This conditional modifier tests whether result is equal to or less than zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.o	Overflow	**OF.** This conditional modifier tests whether the computed result causes overflow – the computed result is outside the range of the destination data type. Note: The legacy condition modifier behavior is different from IEEE exception Overflow flag. For inf float to int conversion, .o will set the legacy Overflow flag, but IEEE exception Overflow flag won't be set. All other internal conditional signals are ignored.
.u	Unordered	**NC.** This conditional modifier tests whether the computed result is a NaN (unordered). All other internal conditional signals are ignored.

Destination Hazard

GEN architecture has built-in hardware to avoid destination hazard.

Destination Hazard stands for the risk condition when multiple operations are trying to write to the same destination and the result of the destination may be ambiguous. This may or may not happen on GEN for two instructions with the same destination, or with destinations that have overlapped register region, depending on the ordering of the arrival of destination results. Let's consider two instructions in a thread with potential destination hazard. There may be other instruction between them as long as there is no instruction sourcing the same destination. Using register scoreboards, GEN hardware automatically takes care of the destination hazard by not issuing the second instruction until the destination scoreboard is cleared. However, for certain cases, in fact for most cases, such destination hazard indicated by the register scoreboard is false, causing unnecessary delay of instruction issuing. This may result in lower performance. The destination dependency control field in the instruction word {*NoDDClr, NoDDChk*} allows software to selectively override such hardware destination dependency mechanism. Such performance optimization hooks must be used with extreme caution. When it is not certain that it is a false destination hazard, the programmer should rely on hardware to resolve the dependency.

As the destination dependency control field does not apply to *send* instruction, there is only one condition that a programmer may use the {*NoDDClr, NoDDChk*} capability.

Description
Instructions other than send, may use this control as long as operations that have different pipeline latencies are not mixed. The operations that have longer latencies are: ▪ Opcodes *pln*, *lrp*, *dp**. ▪ Operations involving double precision computation. ▪ Integer DW multiplication where both source operands are DWs.

Non-present Operands

Some instructions do not have two source operands and one destination operand. If an operand is not present for an instruction the operand field in the binary instruction must be filed with null. Otherwise, results are unpredictable.

Specifically, for instructions with a single source, it only uses the first source operand src0. In this case, the second source operand src1 must be set to null and also with the same type as the first source operand src0. It is a special case when src0 is an immediate, as an immediate src0 uses DW3 of the instruction word, which is normally used by src1. In this case, src1 must be programmed with register file ARF and the same data type as src0.

Instruction Prefetch

Due to prefetch of the instruction stream, the EUs may attempt to access up to 8 instructions (512b) beyond the end of the kernel program – possibly into the next memory page. Although these instructions will not be executed, they must be accounted for the prefetch in order to avoid invalid page access faults. GFX software is required to pad the end of all kernel programs with 512b data. A more efficient approach would be to ensure that the page after all kernel programs is at least valid (even if mapped to a dummy page). Note that the **General State Access Upper Bound** field of the STATE_BASE_ADDRESS command can be used to prevent memory accesses past the end of the General State heap (where kernel programs must reside).

ISA Introduction

This chapter contains these sections that introduce this volume.

- Introducing the Execution Unit
- EU Terms and Acronyms
- EU Changes by Processor Generation
- EU Notation

Subsequent chapters cover:

- EU Data Types
- Execution Environment
- Exceptions
- Instruction Set Summary
- Instruction Set Reference
- EU Programming Guide

The EU Programming Guide provides some useful examples and information but is not a complete or comprehensive programming guide.

Introducing the Execution Unit

This section introduces the Execution Unit (EU), a simple and capable processor within the GPU that supports graphics processing within the graphics pipelines, can do general purpose computing (GPGPU), and responds to exceptional conditions via the System Routine.

The EU provides parallelism at two levels: thread and data element. Multiple threads can execute on the EU; the number executing concurrently depends on the processor and is transparent to EU code. Each thread has its own registers (GRF and ARF, described below). Most EU instructions operate on arrays of data elements; the number of data elements is normally the *ExecSize (execution size)* or number of channels for the instruction. A *channel* is a logical unit of execution for data element access, masking, and flow control within instructions. The number of channels is independent of the number of physical ALUs or FPUs for a particular graphics processor.

EU native instructions are 128 bits (16 bytes) wide. Some combinations of instruction options can use compact instruction formats that are 64 bits (8 bytes) wide. Identifying instructions that can be compacted and creating the compact representations is done by software tools, including compilers and assemblers.

Data manipulation instructions have a destination operand (dst) and one, two, or three source operands (src0, src1, or src2). The instruction opcode determines the number of source operands. An instruction's last source operand can be an immediate value rather than a register.

Data read or written by a thread is generally in the thread's *GRF (General Register File)*, 128 general registers, each 32 bytes. A data element address within the GRF is denoted by a register number (r0 to r127) and a subregister number. In the instruction syntax, subregister numbers are in units of data element size. For example, a :d (Signed Doubleword Integer) element can be in subregister 0 to 7, corresponding to byte numbers in the instruction encoding of 0, 4, ... 28.

The EU cannot directly read or write data in system memory.

Specialized registers used to implement the ISA are in a distinct per thread *Architecture Register File (ARF)*. Each such register or group of related registers has its own distinct name. For example, ip is the instruction pointer and f0 is a flags register. An ARF register can be a src0 or dst operand but not a src1 or src2 operand. There are restrictions on how particular ARF registers are accessed that should be understood before directly reading or writing those registers. See the ARF Registers section for more information.

The EU supports both integer and floating-point data types, as described in the Numeric Data Types section.

For EU flow control, each channel has its own per-channel instruction pointer (PcIP[n]) and only executes an instruction when IP == PcIP[n] and any other masks enable the channel. Most flow control instructions use signed offsets from the current instruction address to reference their targets. Unconditional branches are done using *mov* with IP as the destination. Flow control can also use SPF (Single Program Flow) mode to execute with a single instruction pointer (IP).

The EU ISA supports predication, masking, regioning, swizzling, some type conversions, source modification, saturation, accumulator updates, and flag updates as part of instruction execution:

- *Predication* creates a bit mask (*PMask*) to enable or disable channels for a particular instruction execution. Pmask is derived from flag register and subregister values using boolean formulas determined by the PredCtrl (Predicate Control) and PredInv (Predicate Inversion) instruction fields. See the Predication section.
- *Masking* is the overall process of determining which channels execute for a given instruction based on five factors:
  - Number of channels (only channels in [0, *ExecSize* - 1] can execute)
  - Execution mask (*EMask*)
  - Whether the channel is on the instruction (if not in Single Program Flow mode and MaskCtrl is not NoMask)
  - Predicate mask (*PMask*)
  - In Align16 mode, any enabling of channels using the Dst.ChanEn instruction field (if MaskCtrl is not NoMask).
- *Regioning* specifies an array of data elements contained in one or two registers, with options for scattering, interleaving, or repeating data elements in registers using width and stride values, subject to significant constraints. Regioning also includes access mode (Align1 or Align16) and addressing mode (Direct or Indirect). See the Registers and Register Regions section.
- *Swizzling* allows small scale reordering of data elements within groups of four at the input using the modulo 4 channel names x, y, z, and w. For example, a swizzle of .wzyx with an *ExecSize* of 8 reads execution channels 0 to 7 from these input channels: 3, 2, 1, 0, 7, 6, 5, and 4. Swizzling is only available in the Align16 access mode, described in the Execution Environment chapter.
- *Type Conversions* do any needed conversion from source data type to execution data type and from execution data type to destination data type. See Execution Data Type for more information. Each instruction description indicates what combinations of data types are supported.
- *Source Modification* modifies a source operand just before doing the requested operation. For a numeric operation, the choices are:
  - No modification (normal).
  - - indicating negation.
  - (abs) indicating absolute value.
  - -(abs) indicating a forced negative value.

  Source modification logically occurs after any conversion from source data type to execution data type. Each instruction description indicates whether it supports source modification.
- *Saturation* clamps result values to the nearest value within a saturation range determined by the destination type. For a floating-point type, the saturation range is [0.0, 1.0]. For an integer type, the saturation range is the entire range for that type, for example [0, 65535] for the UW (Unsigned Word) type. Each instruction description indicates whether it supports saturation.
- *Accumulator Updates* optionally update the accumulator register or registers in the ARF with destination values as a side effect of instruction execution. The AccWrCtrl instruction field enables accumulator updates. The Accumulator Disable flag in control register 0 (cr0) can be used to

disable accumulator updates, regardless of AccWrCtrl values; for example, this flag may be used in the System Routine.

- *Flag Updates* optionally update a flags register and subregister (f0.0, f0.1, f1.0, or f1.1) with conditional flags based on the CondModifier (Condition Modifier) instruction field. For example, a CondModifier of **.nz** (not zero) assigns flag bits based on whether result elements are not zero (1) or zero (0). Each instruction description indicates whether it supports the Condition Modifier and any restrictions on the values supported.

The EU is not required to execute steps in its internal pipeline sequentially or in order, so long as it produces correct results.

The assembler syntax uses spaces between operands and encloses ExecSize and any predicate in parentheses. Instruction mnemonics, register names, conditional modifiers, predicate controls, and type designators use lowercase. Function names used with the math instruction are UPPERCASE.

```
( pred ) inst cmod sat ( exec_size ) dst src0 src1 { inst_opt, ... }
```

General register destination regions use the syntax r*m.n*<*HorzStride*>:type. General register directly addressed source regions use the syntax r*m.n*<*VertStride*;*Width*,*HorzStride*>:type. You need to understand more about register regioning to understand all of these terms.

The following example assembly language instruction adds two packed 16-element single-precision Float arrays in r4/r5 and r2/r3 writing results to r0/r1, only on those channels enabled by the predicate in f0.0 along with any other applicable masks.

```
(f0.0) add (16) r0.0<1>:f r2.0<8;8,1>:f r4.0<8;8,1>:f
```

EU Terms and Acronyms

This section provides three tables describing EU general terms and acronyms, EU data types, and EU selected ARF registers.

EU General Terms and Acronyms

Term	Description
ALT mode	A floating-point execution mode that maps +/- inf to +/- fmax, +/- denorm to +/-0, and NaN to +0 at the FPU inputs and never produces infinities, denormals, or NaN values as outputs. See IEEE mode.
ALU	Arithmetic Logic Unit. A functional block that performs integer arithmetic and logic operations, as distinct from instruction fetch and decode, floating-point operations (see FPU), or messaging.
AOS	Array Of Structures. Also see SOA.
ARF	Architecture Register File, a distinct register file containing registers used to implement specific ISA features. For example the Instruction Pointer and condition flags are in ARF registers. See GRF.
byte	An 8-bit value aligned on an 8-bit boundary and the basic unit of addressing. Bits within a byte are denoted 0 to 7 from LSB to MSB.

Term	Description
channel	A logical unit of SIMD data parallel execution within a thread and within the EU. The number of physical ALUs or FPUs is not directly related to the number of channels. Supports up to 32 channels.
compact instruction	A 64-bit instruction encoded as described in the EU Compact Instructions section. Only some combinations of instruction parameters can be encoded as compact instructions. See native instruction.
compressed instruction	An instruction that writes to two destination registers. For example a SIMD16 instruction with Float operands can write channels 0 to 7 to one 32-byte general register and channels 8 to 15 to a second, consecutive 32-byte general register.
denorm	A very small but nonzero number in IEEE mode, with a magnitude less than the smallest normalized floating-point number representable in a particular floating-point format. Denormals lose precision as their values approach zero, called *gradual underflow*.
DWord	Doubleword. A 32-bit (4-byte) value aligned on a 32-bit (4-byte) boundary. Bits within a DWord are denoted 0 to 31 from LSB to MSB.
EOT	End of Thread. A flag set on a *send* or *sendc* instruction to terminate a thread's execution on the EU.
EU	Execution Unit. The single GPU unit described in this volume. This volume describes individual data parallel execution paths within a thread in the EU as *channels*. A few fields, like EUID, use EU to refer to a particular hardware resource used to implement the overall EU.
exception	An error or interrupt condition that arises during execution that may transfer control to the System Routine. Some exceptions can be disabled, preventing such transfers. As defined in this volume, some errors do not produce exceptions.
ExecSize	The number of execution channels for a particular instruction. Channels within that number are enabled or disabled by various masks.
floating-point	Numeric types that allow fractional values and often a wider range than integer types. The EU supports binary floating-point types including the single precision type and the double precision type defined by the IEEE 754 standard.
GEN	GEN is sometimes used to refer to Intel's mainstream GPU architecture integrated with recent CPU generations including SNB (Sandy Bridge) and IVB (Ivy Bridge).
GRF	General Register File, a distinct register file containing 128 general registers, r0 to r127. Each general register is 256 bits (32 bytes), can contain any type of data, and can be accessed with any valid combination of addressing mode, access mode, and region parameters. A general register is directly addressed using a register number and subregister number, or indirectly addressed using an address subregister (index register) and an address immediate offset.
IEEE mode	A floating-point execution mode that supports all the kinds of floating-point values described by the IEEE 754 standard: normalized finite nonzero binary floating-point numbers, signed zeros, signed infinities, signed denormals that are closer to zero than any normalized value but still nonzero, and NaN (not a number) values. See ALT mode.
index register	An address subregister when used for indirect addressing.
inf	Infinity, +inf or -inf, as a floating-point value in IEEE mode.
instruction	In this volume, *instruction* always refers to an EU instruction.

Term	Description
ISA	Instruction Set Architecture, processor aspects visible to programs and programmers and independent of a particular implementation, including data types, registers, memory access, addressing modes, exceptions, instruction encodings, and the instruction set itself. An ISA does not include instruction timing, hardware pipeline details, or the number of physical resources (ALUs, FPUs, instruction decoders) mapped to logical constructs (threads, channels). This volume also includes a recommended assembly language syntax, closely related to the ISA but logically distinct from it.
LSB	Least significant bit.
message	A data structure transmitted from a thread to another thread, to a shared function, or to a fixed function. Message passing is the primary communication mechanism of the GEN architecture.
MSB	Most significant bit.
NaN	Not a Number. A non-numeric value allowed in the standard single precision and double precision floating-point number formats. Quiet NaNs propagate through calculations and signaling NaNs cause exceptions. NaNs are not used in the ALT floating-point mode.
native instruction	A 128-bit instruction, the regular instruction format that allows all defined instruction parameters and options. Some instructions can also be encoded using a 64-bit compact instruction format.
OWord	Octword. A 128-bit (16-byte) value aligned on a 128-bit (16-byte) boundary. Bits within an OWord are denoted 0 to 127 from LSB to MSB. This term is used rarely and may be dropped from future versions of this volume.
packed	A register region is described as *packed* if its elements are adjacent in memory, with no intervening space, no overlap, and no replicated values. If there is more than one element in a row, elements must be adjacent. If there is more than one row, rows must be adjacent. When two registers are used, the registers must be adjacent and both must exist. The immediate vector data types are all described as *Packed* because each such type packs several small data elements into a 32-bit immediate value.
QWord	Quadword. A 64-bit (8-byte) value aligned on a 64-bit (8-byte) boundary. Bits within a QWord are denoted 0 to 63 from LSB to MSB.
region	A collection of data locations in registers and subregisters for a source or destination operand. The associated regioning parameters allow regions to be arrays with various layouts.
register	Part of the directly accessible state of an EU program, such as a general register in the GRF or an architecture register in the ARF. Note that system memory is not directly accessible.
SIMD	Single Instruction Multiple Data. Each EU instruction can operate on multiple data elements in parallel, as specified by the instruction's ExecSize.
SIP	System Instruction Pointer, the starting IP value for the System Routine.
SOA	Structure of Arrays. Also see AOS.
SPF	Single Program Flow. A mode in which every execution channel uses the common instruction pointer, IP in the ip register. The SPF bit in the control register is 1 to enable SPF and 0 to disable it. If SPF is disabled, then each execution channel n has its own instruction pointer, PcIP[n] and each channel n is only eligible to execute, subject to other masking, when PcIP[n] == IP.

Term	Description
swizzle	Rearrange data elements within a vector. The EU supports modulo four swizzling of register source operands at the input in the Align16 access mode.
System Routine	A global EU exception handling routine. Any enabled exception from any EU thread transfers control to this routine.
thread	An instance of a program executing on the EU. The life cycle for a thread on the EU starts with the first instruction after being dispatched to the EU by the Thread Dispatcher and ends after executing a *send* or *sendc* instruction with EOT set, signaling thread termination. Threads can be independent or can communicate with each other via the Message Gateway shared function.
word	A 16-bit (2-byte) value aligned on a 16-bit (2-byte) boundary. Bits within a word are denoted 0 to 15 from LSB to MSB. *Word* has denoted a 16-bit unit for Intel processors since the 8086 and 8088 processors were introduced in 1978.

The next table lists all EU numeric data types. See the Numeric Data Types section for more information about each data type.

EU Numeric Data Types (Listed Alphabetically by Short Name)

Short Name	Assembler Syntax	Long Name	Size in Bytes	Size in Bits	Integral or Float	Description
B	:b	Signed Byte Integer	1	8	I	Signed integer in the range -128 to 127.
D	:d	Signed Doubleword Integer	4	32	I	Signed integer in the range -2^{31} to $2^{31} - 1$.
DF	:df	Double Float	8	64	F	Double precision floating-point number.
F	:f	Float	4	32	F	Single precision floating-point number.
HF	:hf	Half Float	2	16	F	Half precision floating-point number.
Q	:q	Signed Quadword Integer	8	64	I	Signed integer in the range -2^{63} to $2^{63} - 1$.
UB	:ub	Unsigned Byte Integer	1	8	I	Unsigned integer in the range 0 to 255.
UD	:ud	Unsigned Doubleword Integer	4	32	I	Unsigned integer in the range 0 to $2^{32} - 1$.
UQ	:uq	Unsigned Quadword Integer	8	64	I	Unsigned integer in the range 0 to $2^{64} - 1$.
UV	:uv	Packed Unsigned Half Byte Integer Vector	4	32	I	Eight 4-bit unsigned integer values each in the range 0 to 15. Only used as an immediate value.
UW	:uw	Unsigned Word Integer	2	16	I	Unsigned integer in the range 0 to 65,535.
V	:v	Packed Signed Half Byte Integer Vector	4	32	I	Eight 4-bit signed integer values each in the range -8 to 7. Only used as an immediate value.
VF	:vf	Packed Restricted Float	4	32	F	Four 8-bit restricted float values. Only used

Short Name	Assembler Syntax	Long Name	Size in Bytes	Size in Bits	Integral or Float	Description
		Vector				as an immediate value.
W	:w	Signed Word Integer	2	16	I	Signed integer in the range -32,768 to 32,767.

The next table lists the seven ARF registers that you should understand first, omitting several others. See the ARF Registers section for more information, including descriptions of additional registers not listed below.

EU Selected ARF Registers (Listed Alphabetically by Name)

Name	Assembler Syntax	Description
Accumulators	acc0, acc1	Data registers that can hold integer or floating-point values of various sizes. Many instructions can implicitly update accumulators with a copy of destination values, done by setting the AccWrCtrl instruction option. A few instructions, like *mac* (Multiply Accumulate), use the accumulators as an implicit source operand, useful for some iterative calculations. There are added accumulator registers acc2 to acc9 for special purposes; these added accumulators are not generally used.
Address Register	a0.*s*	Holds subregisters primarily used for indirect addressing. Each subregister is a 16-bit UW (Unsigned Word) value. For an indirectly addressed operand or element, the subregister value plus an AddrImm signed offset field determines the byte address (RegNum and SubRegNum) within the register file (GRF). There are 16 address subregisters.
Control Register	cr0.*s*	Contains bit fields for floating-point modes, flow control modes, and exception enable/disable. Also contains exception indicator flags and saves the AIP (Application Instruction Pointer) on transferring control to the System Routine to handle an exception.
Flags	f*r.s*	Used as the outputs for various channel conditional signals, such as equality/zero or overflow. Used as the inputs for predication. There are two 32-bit flags registers each containing two 16-bit subregisters.
Instruction Pointer (IP)	ip	References the current instruction in memory, as an unsigned offset from the General State Base Address. IP is the thread's overall instruction pointer. Each channel n can have its own instruction pointer (PcIP[n]). If not in Single Program Flow mode (SPF is 0) then only those channels where PcIP[n] == IP are eligible to execute the instruction, if enabled by all other applicable masks.
Null Register	null	Indicates a non-existent operand. Unused operands in the instruction format, like the

Name	Assembler Syntax	Description
		unused second source operand field in a *mov* instruction, are encoded as null. For present source operands, reading a null source operand returns undefined values. For null destination operands, results are discarded but any implicit updates to accumulators or flags still occur.
State Register	sr0.*s*	Contains thread identification and scheduling fields, and mask fields for enabling or disabling channels.

Execution Units (EUs)

Each EU is a vector machine capable of performing a given operation on as many as 16 pieces of data of the same type in parallel (though not necessarily on the same instant in time). In addition, each EU can support a number of execution contexts called *threads* that are used to avoid stalling the EU during a high-latency operation (external to the EU) by providing an opportunity for the EU to switch to a completely different workload with minimal latency while waiting for the high-latency operation to complete.

For example, if a program executing on an EU requires a texture read by the sampling engine, the EU may not necessarily idle while the data is fetched from memory, arranged, filtered and returned to the EU. Instead the EU will likely switch execution to another (unrelated) thread associated with that EU. If that thread encounters a stall, the EU may switch to yet another thread and so on. Once the Sampler result arrives back at the EU, the EU can switch back to the original thread and use the returned data as it continues execution of that thread.

The fact that there are multiple EU cores each with multiple threads can generally be ignored by software. There are some exceptions to this rule: e.g., for:

Description
thread-to-thread communication (see *Message Gateway*, *Media*)
synchronization of thread output to memory buffers (see *Geometry Shader*)

In contrast, the internal SIMD aspects of the EU are very much exposed to software.

This volume will not deal with the details of the EUs.

EU Changes by Processor Generation

This section describes how the EU changes for particular processor generations. Instruction compaction tables can differ for each generation, so that is not mentioned in these lists. Particular readers and audiences can see only certain content in this section. Workarounds for particular generations, SKUs, or steppings are not included in these lists. Some small changes in instruction layouts are not included in these lists.

Description
These features or behaviors are added for IVB+, continuing to later generations: • The maximum *ExecSize* increases to 32, for byte or word operands. • Increase the number of flag registers from one to two. • Add the NibCtrl field, used with QtrCtrl to select groups of channels or flags. • Add the DF (Double Float) data type, the first time an 8-byte data type is supported. DF only supports the IEEE floating-point mode and not the ALT floating-point mode. • Add a shared source data type field and a destination data type field for instructions with three source operands, allowing F (Float), DF (Double Float), D (Signed Doubleword Integer), or UD (Unsigned Doubleword Integer) types to be specified. • Add bit manipulation instructions: *bfi1*, *bfi2*, *bfrev*, *cbit*, *fbh*, and *fbl*. • Add the integer *addc* (Add with Carry) and *subb* (Subtract with Borrow) instructions. • Add the *brc* (Branch Converging) and *brd* (Branch Diverging) instructions. • For the *cmp* and *cmpn* instructions, relax the accumulator restrictions. • For the *sel* instruction, remove the accumulator restriction. • Add the Rounding Mode and Double Precision Denorm Mode fields in Control Register 0.
These features or behaviors are added for HSW+, continuing to later generations: • DF (Double Float) operands use an element size of 8. Regioning and channel parameters for the DF type are determined normally, in the same way as for other types. • Add the channel enable register, flow control registers, and stack pointer register in the ARF. • In the Control Register, add the Force Exception Status and Control, Context Save Status, and Context Restore Status bits. • Relative instruction offsets (JIP, UIP) are now 32-bit values in units of bytes (rather than 16-bit values using 8-byte units) for some instructions: *brc*, *brd*, *call*, and *jmpi*. • A *call* instruction can get the relative instruction offset (JIP) from a register. • Add the *calla* (Call Absolute) instruction. • A *mov* instruction with different source and destination types can now use conditional modifiers.
These features or behaviors are added for BDW+, continuing to later generations: • Add the HF (Half Float) type and a corresponding HF execution data type and execution path. • Add flags to indicate IEEE floating-point exceptions and to enable or disable exception reporting to those flags. • Add the Single Precision Denorm Mode bit in Control Register 0. It can be enabled to allow calculations using the F (Float) type in IEEE floating-point mode to support denormals and gradual underflow. • Add the Q (Signed Quadword Integer) and UQ (Unsigned Quadword Integer) types. Integer source types cannot mix 64-bit and 8-bit operands. Some integer instructions (e.g., *avg*) do not support Q or UQ source types. • Instructions with one source operand and a 64-bit source type can have immediate 64-bit source operands.

Description
• The JIP and UIP relative instruction offset fields in all remaining flow control instructions (those instructions that did not make this change for HSW+) are 32-bit values in units of bytes (rather than 16-bit values using 8-byte units). • The instruction layout is noticeably different. The SrcType and DstType instruction fields are widened to allow for more type encodings as three types are added. The AddrSubRegNum instruction field is widened to allow for 16 address subregisters rather than 8. The layout now supports 64-bit immediate source operands for one-source instructions and 32-bit relative instruction offset fields for flow control instructions. • In the 3-source instruction format, widen the SrcType and DstType fields and add an encoding for the HF (Half Float) type. • Add a compact instruction format for 3-source instructions. • Use a different source modifier interpretation for logical (*and*, *not*, *or*, *xor*) instructions. • An accumulator source operand for a logical instruction can now have a source modifier. • Add eight address subregisters, increasing the number of address subregisters from 8 to 16. • For the *brc* and *brd* instructions do not allow the **Switch** instruction option. • For the *cmp* and *cmpn* instructions, remove the accumulator restrictions. • Add the *goto* instruction, reusing the opcode for the discontinued SNB *fork* instruction. • Add the *join* instruction. • For the *lzd* instruction, remove the accumulator restriction. • The *mach* instruction reverses the roles of the two source operands compared to previous generations. • Add the *madm* instruction. • Enhance the *math* instruction to allow some immediate source values and support the INVM and RSQRTM functions. • For the *mul* instruction, relax the accumulator restriction on source operands so it applies for only integer source operands. • For the rounding instructions (*rndd*, *rnde*, *rndu*, and *rndz*), remove the accumulator restrictions. • Revise the *shl* and *shr* instructions to use the low 6 bits of the shift count in QWord mode, versus the low 5 bits otherwise. • Add the *smov* instruction. • Add eight accumulator registers, acc2 to acc9, used only for the special purpose of emulating IEEE-compliant fdiv and sqrt operations. • Add message control registers. • Widen the sp (Stack Pointer) register to 64 bits. • Add the IEEE Exception, Page Fault Status, and Page Fault Code bit fields in the State Register. • Remove some regioning restrictions when operands span two registers.
These features or behaviors are added for SKL+, continuing to later generations: • Add the Half Precision Denorm Mode bit in the Control Register. It can be enabled to allow calculations using the HF (Half Float) type to support denormals and gradual underflow.

EU Notation

The Courier New font is used for code examples and for the Syntax, Format, and Pseudocode sections in the instruction reference.

The *italic* font style is used for instruction mnemonics outside of code (e.g., the *send* instruction), for syntactic production names, for key values in algorithms (*ExecSize*), and to emphasize a word or phrase. For example: When bit 10 is set, the destination register scoreboard is *not* cleared.

The **bold** font weight is used for the short name and long name of a bit field being described, for value names being defined, for syntactic terminals, for unnumbered subheadings, and for the terms Note or Workaround used to introduce a paragraph.

Bit field names and value names used where not being defined and not as syntactic terminals are in plain text.

Bit field values in hex use the 0x prefix. The BSpec currently uses the 0x prefix for hex in some parts and the h suffix for hex in other parts. For single bits, values appear as simply 0 or 1. For multi-bit binary values, the appropriate number of binary digits appears with a b suffix.

Instruction mnemonics are lowercase. Function names invoked using the *math* instruction are UPPERCASE. For example, SQRT.

Device names in the new syntax do not use the Dev prefix or square brackets and often appear in tagged tables with a blue background. For example:

Device names in the old syntax are in plain text in square brackets. For example, .

Tables describing bit field layouts or registers proceed from most significant to least significant bits. Figures showing bit fields or registers show most significant bits on the left and least significant bits on the right.

Any bit, field, or register described as Reserved should be regarded as undefined and unpredictable. Such bits should be treated as follows:

- When testing values, do not depend on the state of reserved bits. Mask out or otherwise ignore such bits.
- Sometimes software must initialize reserved bits. For example, a compiler must write complete instruction values when creating an instruction stream, including reserved bits. In such cases, write reserved bits as zeros unless otherwise indicated.
- Do not use reserved bits as extra storage for software-defined values; put nothing in such bits.
- When saving state and restoring state, save and restore any reserved bits as well.
- Do not assume that reserved bits are invariant between explicit writes. Software should function even if reserved bits change in undefined and unpredictable ways.

Any value, encoding, or combination of values or encodings described as Reserved must not be used. The EU's behavior is undefined in this case.

When a combination of instruction parameters or an EU state is described as producing undefined results or behavior, do not assume that undefined results or behavior are confined to specific instructions, operands, registers, or channels.

Execution Environment

EU Data Types

Fundamental Data Types

Numeric Data Types

Floating Point Modes

- IEEE Floating Point Mode
  - Partial Listing of Honored IEEE 754 Rules
  - Complete Listing of Deviations or Additional Requirements vs IEEE 754
  - Min/Max of Floating Point Numbers
- Alternative Floating Point Mode

Floating-Point Support

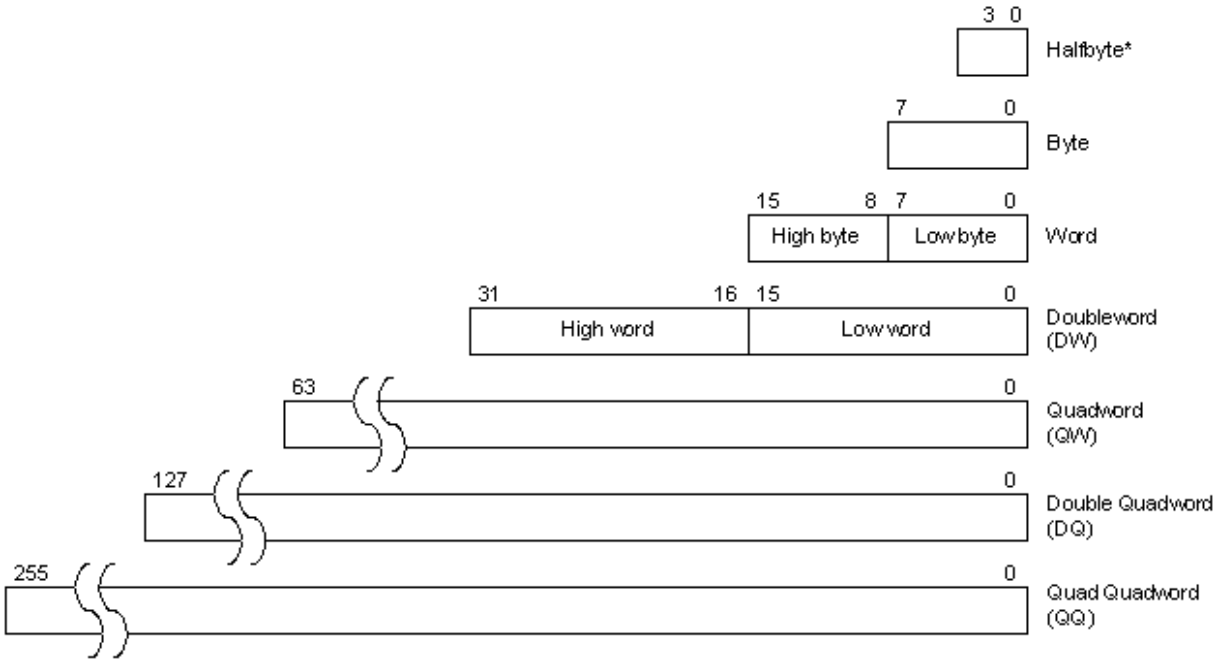
- IEEE Floating-Point Exceptions
- Floating-Point Compare Operations

Type Conversion

Fundamental Data Types

The fundamental data types in the GEN architecture are halfbyte, byte, word, doubleword (DW), quadword (QW), double quadword (DQ) and quad quadword (QQ). They are defined based on the number of bits of the data type, ranging from 4 bits to 256 bits. As shown in the figure below, a halfbyte contains 4 bits, a byte contains 8 bits, a word contains two bytes, a doubleword (DWord) contains two words, and so on. Halfbyte is a special data type that is not accessed directly as a standalone data element; it is only allowed as a subfield of the numeric data type of “packed signed halfbyte integer vector” described in the next section.

Fundamental Data Types



With the exception of halfbyte, the access of a data element to/from a GEN register or to/from memory must be aligned on the natural boundaries of the data type. The natural boundary for a word has an even-numbered address in units of bytes. The natural boundary for a doubleword has an address divisible by 4 bytes. Similarly, the natural boundary for a quadword, double quadword, and quad quadword has an address divisible by 8, 16, and 32 bytes, respectively. Double quadword, and quad quadword do not have corresponding numeric data types. Instead, they are used to describe a group (a vector) of numeric data elements of smaller size aligned to larger natural boundaries.

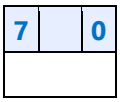
Numeric Data Types

The numeric data types defined in the GEN architecture include signed and unsigned integers and floating-point numbers (floats) of various sizes. These numeric data types are described below.

Integer Numeric Data Types

The Execution Unit supports the following integer data types. Signed integer types use two's complement representation for negative numbers.

UB: Unsigned Byte, 8-bit Unsigned Integer



B: Byte, 8-bit Signed Integer

7	6		0
S			

UW: Unsigned Word, 16-bit Unsigned Integer

15		0

W: Word, 16-bit Signed Integer

15	14		0
S			

UD: Unsigned Doubleword, 32-bit Unsigned Integer

31		0

D: Doubleword, 32-bit Signed Integer

31	30		0
S			

UQ: Unsigned Quadword, 64-bit Unsigned Integer BDW+

63		0

Q: Quadword, 64-bit Signed Integer BDW+

63	62		0
S			

UV: Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0

V: Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer

31	28	27	24	23	20	19	16	15	12	11	8	7	4	3	0
S		S		S		S		S		S		S		S	

The following table summarizes the EU integer data types.

Execution Unit Integer Data Types

Notation	Size in Bits	Name	Range
UB	8	Unsigned Byte Integer	[0, 255]
B	8	Signed Byte Integer	[-128, 127]
UW	16	Unsigned Word Integer	[0, 65535]
W	16	Signed Word Integer	[-32768, 32767]
UD	32	Unsigned Doubleword Integer	$[0, 2^{32} - 1]$
D	32	Signed Doubleword Integer	$[-2^{31}, 2^{31} - 1]$
UQ	64	Unsigned Quadword Integer	$[0, 2^{64} - 1]$
Q	64	Signed Quadword Integer	$[-2^{63}, 2^{63} - 1]$
UV	32	Packed Unsigned Half-Byte Integer Vector	[0, 15] in each of eight 4-bit immediate vector elements.
V	32	Packed Signed Half-Byte Integer Vector	[-8, 7] in each of eight 4-bit immediate vector elements.

Restriction: Only a raw move using the *mov* instruction supports a packed byte destination register region. For information about raw moves, refer to the **Description** in *mov – Move*.

Floating-Point Numeric Data Types

The Execution Unit supports the following floating-point data types.

Type or Description
The Float type uses the single precision format specified in IEEE Standard 754-1985 for Binary Floating-Point Arithmetic.
The Double Float type uses the double precision format specified in IEEE Standard 754-1985 for Binary Floating-Point Arithmetic.
In the ALT floating-point mode, representations for infinities, denorms, and NaNs within those formats are not used.

Type or Description
The EU does not support the double extended precision (80-bit) floating-point format found in the x86/x87/Intel 64 floating-point registers. All floating-point formats are signed using signed magnitude representation (a distinct sign bit, separate from the magnitude information).
The Half Float type uses the binary16 format specified in IEEE Standard 754-2008.
The F (Float) type supports both the ALT and IEEE floating-point modes, controlled by the Single Precision Floating-Point Mode bit in the Control Register.
Whether IEEE mode F calculations support denorms or flush denormalized values to zero is controlled by the Single Precision Denorm Mode bit in the Control Register.
The DF (Double Float) type only supports the IEEE floating-point mode. Whether DF calculations support denorms or flush denormalized values to zero is controlled by the Double Precision Denorm Mode bit in the Control Register.
The HF (Half Float) type only supports the IEEE floating-point mode.
Whether HF calculations support denorms or flush denormalized values to zero is controlled by the Half Precision Denorm Mode bit.

HF: Half Float, 16-bit Half-Precision Floating-Point Number BDW+

15	14		10	9		0
S	biased exp.			fraction		

F: Float, 32-bit Single-Precision Floating-Point Number

31	30		23	22		0
S	biased exponent			fraction		

DF: Double Float, 64-bit Double-Precision Floating-Point Number

63	62		52	51		0
S	biased exponent			fraction		

VF: Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number

31	30		28	27		24	23	22		20	19		16	15	14		12	11		8	7	6		4	3		0
S	b. exp.			frac.			S	b. exp.			frac.			S	b. exp.			frac.			S	b. exp.			frac.		

The following table summarizes the EU floating-point data types.

Execution Unit Floating-Point Data Types

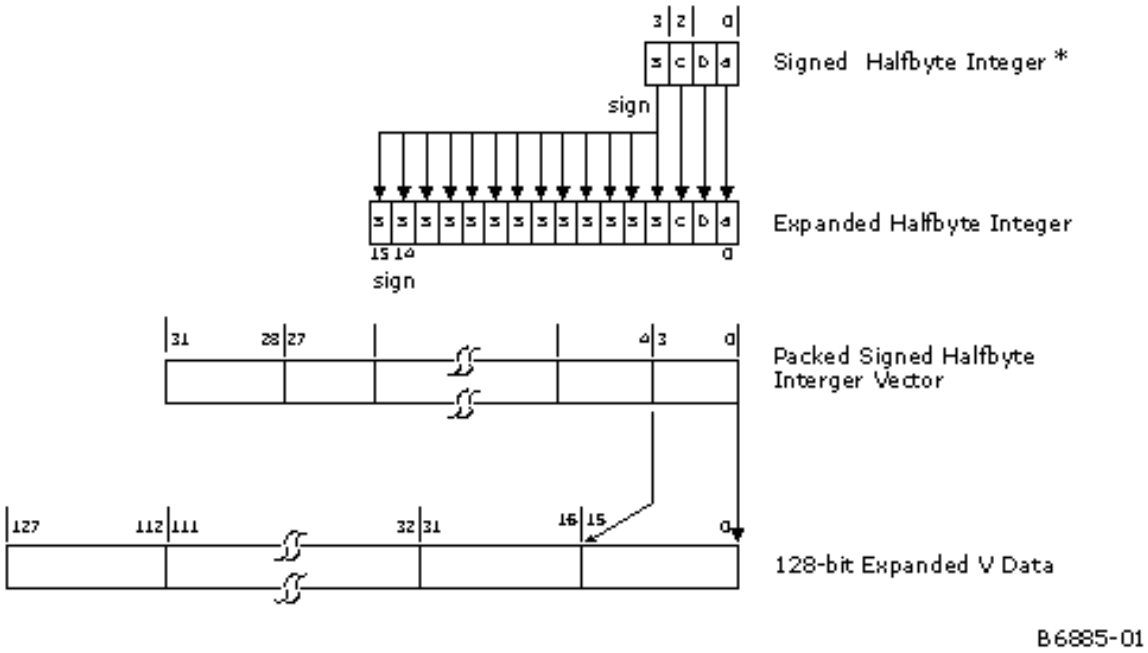
Notation	Size in Bits	Name	Range
HF	16	Half Float	Half precision, 1 sign bit, 5 bits for the biased exponent, and 10 bits for the significand: $[-(2-2^{-10})^{31}...-2^{-40}, 0.0, 2^{-40}... (2-2^{-10})^{31}]$
F	32	Float	Single precision, 1 sign bit, 8 bits for the biased exponent, and 23 bits for the significand: $[-(2-2^{-23})^{127}...-2^{-149}, 0.0, 2^{-149}... (2-2^{-23})^{127}]$
DF	64	Double Float	Double precision, 1 sign bit, 11 bits for the biased exponent, and 52 bits for the significand: $[-(2-2^{-52})^{1023}...-2^{-1074}, 0.0, 2^{-1074}... (2-2^{-52})^{1023}]$
VF	32	Packed Restricted Float Vector	Restricted precision. Each of four 8-bit immediate vector elements has 1 sign bit, 3 bits for the biased exponent (bias of 3), and 4 bits for the significand: [–31... –0.1328125, -0, 0, 0.1328125 ... 31]

Packed Signed Half-Byte Integer Vector

A packed signed halfbyte integer vector consists of 8 signed halfbyte integers contained in a doubleword. Each signed halfbyte integer element has a range from -8 to 7 with the sign on bit 3. This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand or a non-immediate source operand. GEN hardware converts the vector into an 8-element signed word vector by sign extension. This is illustrated in *Numeric Data Types*.

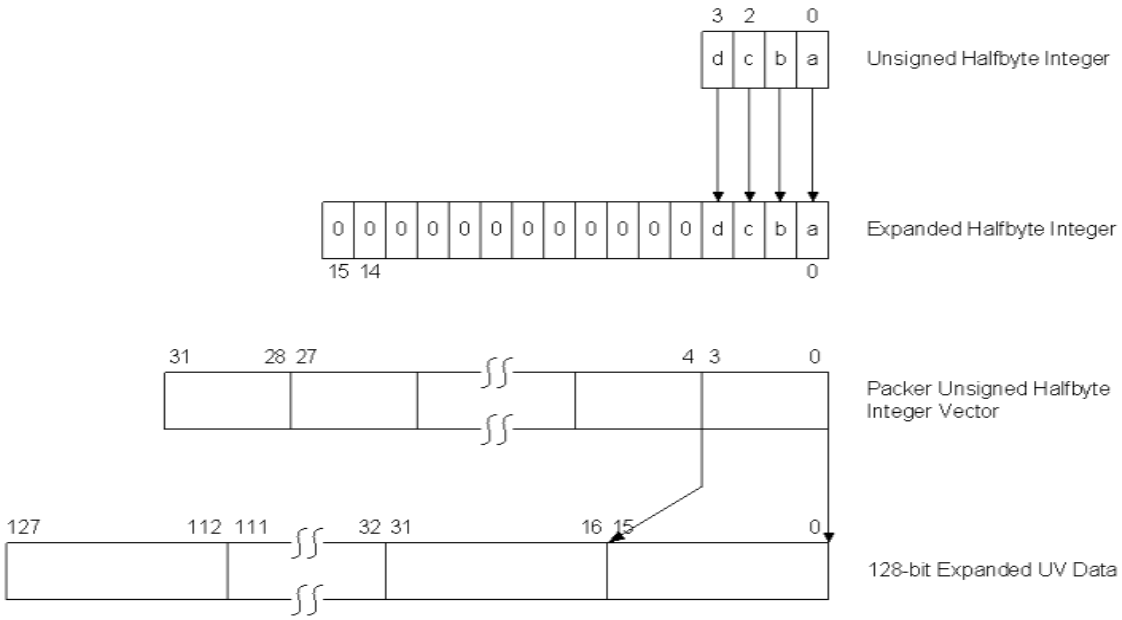
The short hand format notation for a packed signed half-byte vector is **V**.

Converting a Packed Half-Byte Vector to a 128-bit Signed Integer Vector



Packed UnSigned Half-Byte Integer Vector

A packed unsigned halfbyte integer vector consists of 8 unsigned halfbyte integers contained in a doubleword. Each unsigned halfbyte integer element has a range from 0 to 15. This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand or a non-immediate source operand. GEN hardware converts the vector into an 8-element signed word vector.

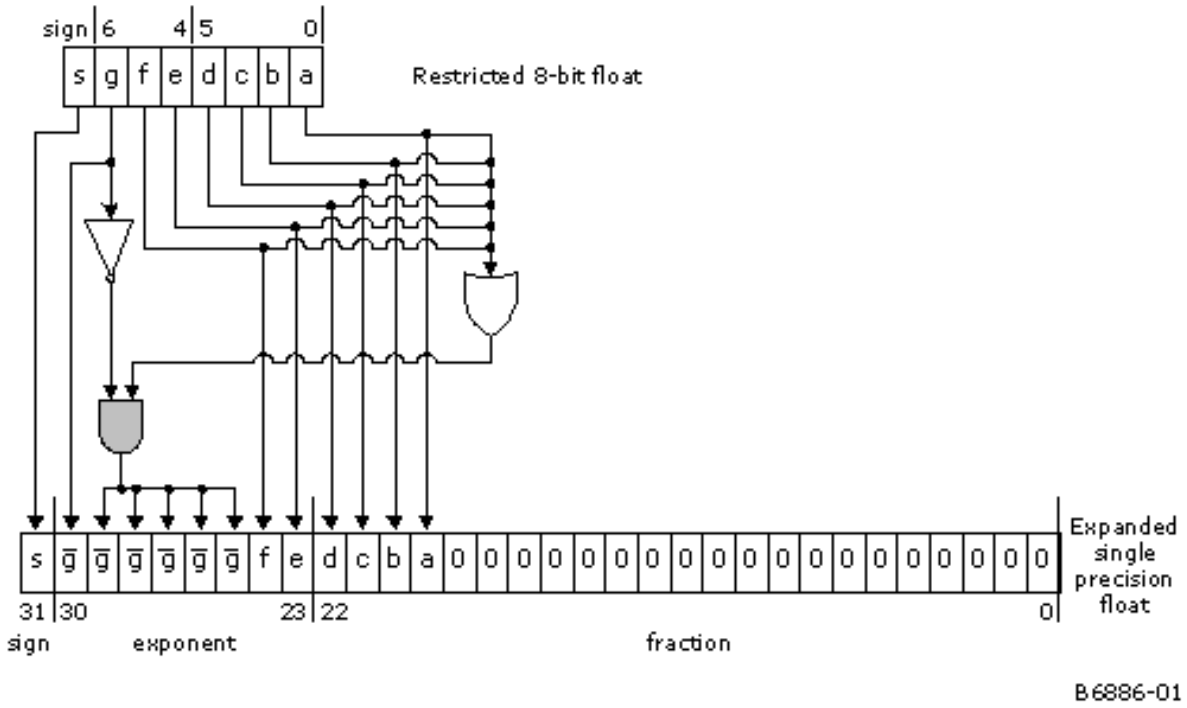


Packed Restricted Float Vector

A packed restricted float vector consists of 4 8-bit restricted floats contained in a doubleword. Each restricted float has the sign at bit 7, a 3-bit coded exponent in bits 4 to 6, a 4-bit fraction in bits 0 to 3, and an implied integer 1. The exponent is in excess-3 format – having a bias of 3. Restricted float provides zero, positive/negative normalized numbers with a small range (3-bit exponent) and small precision (4-bit fraction). This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand, or a non-immediate source operand.

The following figure shows how to convert an 8-bit restricted float into a single precision float. Converting a 3-bit exponent with a bias of 3 to an 8-bit exponent with a bias of 127 is by adding 4, or equivalently copying bit 2 to bit 7 and putting the inverted bit 2 to bits 6:2. A special logic is also needed to take care of positive/negative zeros.

Conversion from a Restricted 8-bit Float to a Single-Precision Float



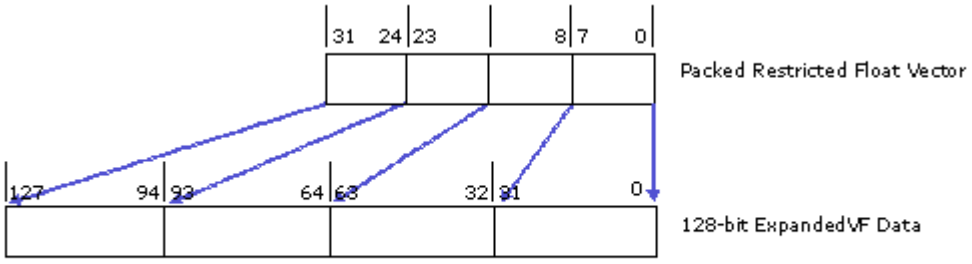
The following table shows all possible numbers of the restricted 8-bit float. Only normalized float numbers can be represented, including positive and negative zero, and positive and negative finite numbers. Normalized infinites, NaN, and denormalized float numbers cannot be represented by this type. It should be noted that this 8-bit floating point format does not follow IEEE-754 convention in describing numbers with small magnitudes. Specifically, when the exponent field is zero and the fraction field is not zero, an implied one is still present instead of taking a denormalized form (without an implied one). This results in a simple implementation but with a smaller dynamic range – the magnitude of the smallest non-zero number is 0.1328125.

Examples of Restricted 8-bit Float Numbers

Class	Hex #	Sign [7]	Exponent [6:4]	Fraction [3:0]	Extended 8-bit Exponent	Floating Number in Decimal
Positive Normalized Float	0x70-0x7F	0	111	0000 ... 1111	1000 0011	16 ... 31
	0x60-0x6F	0	110	0000 ... 1111	1000 0010	8 ... 15.5
	0x50-0x5F	0	101	0000 ... 1111	1000 0001	4 ... 7.75
	0x40-0x4F	0	100	0000 ... 1111	1000 0000	2 ... 3.875
	0x30-0x3F	0	011	0000 ... 1111	0111 1111	1 ... 1.9375
	0x20-0x2F	0	010	0000 ... 1111	0111 1110	0.5 ... 0.96875
	0x10-0x1F	0	001	0000 ... 1111	0111 1101	0.25 ... 0.484375
	0x01-0x0F	0	000	0001 ... 1111	0111 1100	0.1328125 ... 0.2421875
	0x00	0	000	0000	0000 0000	0 (+zero)
Negative Normalized Float	0xF0-0xFF	1	111	0000 ... 1111	1000 0011	-16 ... -31
	0xE0-0xEF	1	110	0000 ... 1111	1000 0010	-8 ... -15.5
	0xD0-0xDF	1	101	0000 ... 1111	1000 0001	-4 ... -7.75
	0xC0-0xCF	1	100	0000 ... 1111	1000 0000	-2 ... -3.875
	0xB0-0xBF	1	011	0000 ... 1111	0111 1111	-1 ... -1.9375
	0xA0-0xAF	1	010	0000 ... 1111	0111 1110	-0.5 ... -0.96875
	0x90-0x9F	1	001	0000 ... 1111	0111 1101	-0.25 ... -0.484375
	0x81-0x8F	1	000	0001 ... 1111	0111 1100	-0.1328125 ... -0.2421875
	0x80	1	000	0000	0000 0000	-0 (-zero)

The following figure shows the conversion of a packed exponent-only float to a 4-element vector of single precision floats.

The shorthand format notation for a packed signed half-byte vector is VF.



Floating Point Modes

GEN architecture supports two floating point operation modes, namely IEEE floating point mode (IEEE mode) and alternative floating point mode (ALT mode). Both modes follow mostly the requirements in IEEE-754 but with different deviations. The deviations will be described in details in later sections. The primary difference between these modes is on the handling of Infs, NaNs and denorms. The IEEE floating point mode may be used to support newer versions of 3D graphics API Shaders and the alternative floating point mode may be used to support early Shader versions. Taking DirectX 3D graphics API Shaders for example, shader models before version 3.0 may use the alternative floating point mode, while version 3.0 and following shader models may use the IEEE floating point mode.

These two modes are supported by all units that perform floating point computations, including GEN execution units, GEN shared functions like Extended Math, the Sampler and the Render Cache color calculator, and fixed functions like VF, Clipper, SF and WIZ. Host software sets floating point mode through the fixed function state descriptors for 3D pipeline and the interface descriptor for media pipeline. Therefore different modes may be associated with different threads running concurrently. Floating point mode control for EU and shared functions are based on the floating point mode field (bit 0) of *cr0* register.

IEEE Floating Point Mode

Partial Listing of Honored IEEE-754 Rules

Here is a summary of expected 32-bit floating point behaviors in GEN architecture. Refer to IEEE-754 for topics not mentioned.

- INF – INF = NaN
- 0 * (+/–)INF = NaN
- 1 / (+INF) = +0 and 1 / (–INF) = –0
    - (+/–)INF / (+/–)INF = NaN as A/B = A * (1/B)
- INV (+0) = RSQ (+0) = +INF, INV (–0) = RSQ (–0) = –INF, and SQRT (–0) = –0
- RSQ (–finite) = SQRT (–finite) = NaN
- LOG (+0) = LOG (–0) = –INF, LOG (–finite) = LOG (–INF) = NaN

- NaN (any OP) any-value = NaN with one exception for min/max mentioned below. Resulting NaN may have different bit pattern than the source NaN.
- Normal comparison with conditional modifier of EQ, GT, GE, LT, LE, when either or both operands is NaN, returns FALSE. Normal comparison of NE, when either or both operands is NaN, returns TRUE.
    - **Note:** Normal comparison is either a **cmp** instruction or an instruction with conditional modifier
- Special comparison **cmpn** with conditional modifier of EQ, GT, GE, LT, LE, when the second source operand is NaN, returns TRUE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns FALSE. **Cmpn** of NE, when the second source operand is NaN, returns FALSE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns TRUE.
    - This is used to support the proposed IEEE-754R rule on **min** or **max** operations. For which, if only one operand is NaN, min and max operations return the other operand as the result.
- Both normal and special comparisons of any non-NaN value against +/– INF return exact result according to the conditional modifier. This is because that infinities are exact representation in the sense that +INF = +INF and –INF = –INF.
    - NaN is unordered in the sense that NaN != NaN.
- IEEE-754 requires floating point operations to produce a result that is the nearest representable value to an infinitely precise result, known as "round to nearest even" (RTNE). 32-bit floating point operations must produce a result that is within 0.5 Unit-Last-Place (0.5 ULP) of the infinitely precise result. This applies to addition, subtraction, and multiplication.
- All arithmetic floating point instructions does Round To Nearest Even at the end of the computation, except the round instructions.

Complete Listing of Deviations or Additional Requirements vs IEEE-754

For a result that cannot be represented precisely by the floating point format, the EU uses rounding to nearest or even to produce a result that is within 0.5 Unit-Last-Place(0.5 ULP) of the infinitely precise result.

Description
The rounding mode is specified by the Rounding Mode field in the Control Register.
Source modifiers are not applied to NAN.
Handle denorms as follows: • When inputs are denorms in mixed mode (one of the source operand is half float and other is single precision float OR source is half float and destination is single precision float), on upconversion, they are non-denorms. So computes in 32b can handle this. • When outputs are denorms, the denorms are retained if there is a format conversion. • Mixed mode operations should always behave like sum of individual operations.

Other information regarding floating-point behaviors:

- NaN input to an operation always produces NaN on output, however the exact bit pattern of the NaN is not required to stay the same (unless the operation is a raw "mov" instruction which does not alter data at all.)
- x*1.0f must always result in x (except denorm flushed and possible bit pattern change for NaN).
- x +/- 0.0f must always result in x (except denorm flushed and possible bit pattern change for NaN). But -0 + 0 = +0.
- Fused operations (such as *mac*, *dp4*, *dp3*, etc.) may produce intermediate results out of 32-bit float range, but whose final results would be within 32-bit float range if intermediate results were kept at greater precision. In this case, implementations are permitted to produce either the correct result, or else ±inf. Thus, compatibility between a fused operation, such as *mac*, with the unfused equivalent, *mul* followed by *add* in this case, is not guaranteed.
- As the accumulator registers have more precision than 32-bit float, any instruction with accumulator as a source/destination operand may produce a different result than that using more general registers, as indicated in this table:

Description
Such an instruction may produce a different result than that using GRF registers.

- API Shader divide operations are implemented as x*(1.0f/y). With the two-step method, x*(1.0f/y), the multiply and the divide each independently operate at the 32-bit floating point precision level (accuracy to 1 ULP).
- See the Type Conversion section for rules on converting to and from float representations.

Min Max of Floating Point Numbers

A special comparison called Compare-NaN is introduced in the GEN architecture to handle the difference of above mentioned floating-point comparison and the rules on supporting MIN/MAX. To compute the MIN or MAX of two floating-point numbers, if one of the numbers is NaN and the other is not, MIN or MAX of the two numbers returns the one that is not NaN. When two numbers are NaN, MIN or MAX of the two numbers returns source1.

When both the sources are NaN inputs, the special case in the section 2.4.8 Floating point Min/Max Operations describe the results.
When one source is SNAN, DX, and IEEE treat the outputs differently. The special case section 2.4.8 Floating point Min/Max operations described the results.

Min and Max is supported by conditional select.

Note even though f0.0 is specified in the instruction, the flag register is not touched by this instruction.

The following tables detail the rules for this special compare-NaN operation for floating-point numbers. Notice that excepting "Not-Equal-To" comparison-NaN, last columns in all other tables have 'T'.

Results of "Less-Than" Comparison-NaN – CMPN.L

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	T
-Fin	F	T/F	T	T	T	T	T	T	T
-denorm	F	F	F	F	F	F	T	T	T
-0	F	F	F	F	T	F	T	T	T
+0	F	F	F	F	F	F	T	T	T
+denorm	F	F	F	F	F	F	T	T	T
+Fin	F	F	F	F	F	F	T/F	T	T
+inf	F	F	F	F	F	F	F	F	T
NaN	F	F	F	F	F	F	F	F	F

Results of "Greater-Than or Equal-To" Comparison-NaN – CMPN.GE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	T
-Fin	T	T/F	F	F	F	F	F	F	T
-denorm	T	T	T	T	T	T	F	F	T
-0	T	T	T	T	F	T	F	F	T
+0	T	T	T	T	T	T	F	F	T
+denorm	T	T	T	T	T	T	F	F	T
+Fin	T	T	T	T	T	T	T/F	F	T
+inf	T	T	T	T	T	T	T	T	T
NaN	F	F	F	F	F	F	F	F	F

Alternative Floating Point Mode

The key characteristics of the alternative floating point mode is that NaN, Inf, and denorm are not expected for an application to pass into the graphics pipeline, and the graphics hardware must not generate NaN, Inf, or denorm as computation result. For example, a result that is larger than the maximum representable floating point number is expected to be flushed to the largest representable floating point number, i.e., +fmax. The fmax has an exponent of 0xFE and a mantissa of all one's, which is the same for IEEE floating point mode.

Note that this mode is applicable ONLY to Single Precision Float datatype.

Description
This also implies that ALT mode is not supported when Single precision datatype is involved in format conversion to double precision or half precision.

Description
ALT_MODE is supported for Single Precision float ONLY. Hence, ALT_MODE is NOT supported in mixed mode operation.

Here is the complete list of the differences of legacy graphics mode from the relaxed IEEE-754 floating point mode.

- Any +/- INF result must be flushed to +/- fmax, instead of being output as +/- INF.
- Extended mathematics functions of log(), rsq(), and sqrt() take the absolute value of the sources before computation to avoid generating INF and NaN results.

Alternative Floating Point Mode shows the support of these differences in various hardware units.

Supported Legacy Float Mode and Impacted Units

IEEE-754 Deviations	VF	Clipper	SF	WIZ	EU	EM	Sampler	RC
Any +/- INF result flushed to +/- fmax	Y	Y	Y	Y	Y	Y	Y	Y
Log, rsq, sqrt take abs() of sources	N/A	N/A	N/A	N/A	N/A	Y	N/A	N/A

Alternative Floating Point Mode shows some of the desired or recommended alternative floating point mode behaviors that do not have hardware design impact. The reasons of not needing special hardware support for these items are also provided. This is based on the compliance requirement that can be found in the DirectX 9 specification: "**Handling of NaNs, Infs, and denorms is undefined. Applications should not pass in such values into the graphics pipeline.**"

Dismissed Legacy Behaviors

Suggested IEEE-754 Deviations	Reason for Dismiss
Mov forces (+/-)INF to (+/-)fmax	(+/-)INF is never present as input
(+/-)INF – (+/-)INF = +/- fmax instead of NaN	(+/-)INF is never present as input
Denorm must be flushed to zero in all cases (including trivial mov and point sampling)	Denorm is never present as input
Anything*0=0 (including NaN*0=0 and INF*0=0)	NaN and INF are never present as input
Except propagated NaN, NaN is never generated	NaN is never present as input and GEN never generates NaN based on rules in the previous table
An input NaN gets propagated excepting (a)-(d)	NaN is never present as input
(a) Rcp (and rsq) of 0 yields fmax	N/A, as it is already covered by the general rule "Any +/- INF result flushed to +/- fmax"
(b) Sampler honors 0/0 = 0 as if (1/0)*0	There is no divide in Sampler
I Rcp (and rsq) of INF yields +/- 0	(+/-)INF is never present as input
(d) Sampler honors INF/INF = 0 as if (1/INF)=0 followed by Anything*0 = 0	There is no divide in Sampler

Floating-Point Support

The following sections describe BDW+ floating-point support relative to the IEEE Standard for Floating-Point Arithmetic, currently IEEE Standard 754-2008. These sections cover binary floating-point arithmetic, as the EU provides no support for decimal floating-point arithmetic.

Note: Hardware alone is usually not fully conformant to the IEEE standard. It requires software functions to supplement the hardware.

Compared to previous generations, BDW+ does the following:

- Adds the HF (Half Float) type and a corresponding HF execution data type and execution path.
- Adds flags to indicate IEEE floating-point exceptions and to enable or disable exception reporting to those flags.
- Adds the Single Precision Denorm Mode bit in Control Register 0. It can be enabled to allow calculations using the F (Float) type in IEEE floating-point mode to support denormals and gradual underflow.

Floating-Point Types and Values

The EU supports 16-bit (HF, Half Float), 32-bit (F, Float), and 64-bit (DF, Double Float) types in the IEEE Standard formats (respectively *binary16*, *binary32*, and *binary64* in IEEE 754-2008). See Floating-Point Numeric Data Types for the layout of the supported floating-point types.

Any bit pattern for a floating-point value corresponds to a value defined by the standard: ± finite (normalized nonzero finite number), ± 0 (signed zero), ± inf (signed infinity), ± denorm (denormalized, very small but nonzero number), or NaN (Not a Number). A NaN can be a signaling NaN (sNaN) or quiet NaN (qNaN).

These operating modes are available for the different floating-point types:

- Half Float uses the IEEE floating-point mode.

Description
Half Float support for denormals and gradual underflow is controlled by the Half Precision Denorm Mode bit.

- Float can use the ALT (Alternative Floating-Point Mode) or the IEEE floating-point mode. In IEEE mode, support for denormals and gradual underflow is controlled by the Single Precision Denorm Mode bit in the Control Register.
- Double Float uses the IEEE floating-point mode. Support for denormals and gradual underflow is controlled by the Double Precision Denorm Mode bit in the Control Register.

Flush to zero is not defined by IEEE Standard 754, but is implementation-specific and required by some APIs (including DirectX), thus the EU ISA for BDW supports either using or flushing Float or Double Float denorms based on the respective Denorm Mode bits.

Specifications in this volume sometimes reference ±fmax, the largest finite magnitude representable in a format, and ±fmin, the smallest normalized nonzero magnitude representable in a format. Calculating

those values uses the extreme exponent values for finite nonzero floating-point values, Emax and Emin below, along with the number of explicit fraction bits (not counting the implicit bit in the significand). The following table provides these values, with the fmax and fmin values generally approximate.

Floating-Point Type Parameters

Type	Exp. Bits	Exp. Bias	Emax	Emin	Explicit Fraction Bits	fmax	fmin
HF	5	15	15	-14	10	65504.0	about 6.1E-5
F	8	127	127	-126	23	about 3.4E38	about 1.18E-38
DF	11	1023	1023	-1022	52	about 1.79E308	about 2.23E-308

Where f is the number of explicit fraction bits, the general formula for fmax is $(2.0 - 2^{-f}) * 2^{Emax}$.

The general formula for fmin is 2^{Emin}.

Not a Number (NaN) Formats

A NaN has a biased exponent field with all bits set (as if encoding an exponent of Emax + 1), a nonzero fraction field (as a zero fraction field with that exponent indicates infinity), and either sign bit.

As specified in IEEE Standard 754-2008, the MSB of the fraction field, what would be the first bit following the binary point in a numeric value, determines a NaN's type:

0 - Signaling NaN (sNaN). The remaining fraction bits cannot all be zero.

1 - Quiet NaN (qNaN). The remaining fraction bits can have any value.

When an sNaN is an input, an operation normally signals the Invalid Operation exception and *quietizes* the NaN, producing the equivalent qNaN value, with MSB set to 1, at the output. Raw moves do not check for NaNs and do not signal exceptions or quietize NaN values.

When QNAN as one of the input to an operation this results in QNAN without raising the exception flag. This silently propagates and the output is the same QNAN as in the input.

Intel specifies the value *qNaN Indefinite* as a quiet NaN with all zeros in the remaining fraction bits, those other than the MSB. This value is useful because it is never produced by quietizing an sNaN, thus qNaN Indefinite may be used to initialize floating-point values that are not otherwise initialized by software, allowing the uninitialized case to be distinguished.

The EU applies numeric source modifiers (**-**, **(abs)**, or **-(abs)**) to NaN source values as well as to other values, possibly changing the sign bit of a NaN value when it is propagated. NaN sign bits are normally don't care values.

Per IEEE Standard 754-2008, a NaN's *payload* is contained in all fraction field bits other than the fraction MSB. Thus in the overall floating-point format, the sign bit, biased exponent, and fraction MSB are not part of the payload. NaN payload values are not affected by quietizing or by source modifiers. As noted above, an sNaN must have a nonzero payload and a qNaN can have any payload.

Floating-Point Rounding Modes

The EU supports the four rounding modes required in IEEE Standard 754 for binary floating-point arithmetic. If the unrounded result of infinite precision and range is exactly representable in the destination format, then that exact result is produced and no exception is signaled. For an unrounded result of infinite precision and range that is not exactly representable in the destination format (an *inexact* result), rounding chooses a numerically adjacent value in the destination format, while signaling the Inexact, Overflow, or Underflow exceptions when appropriate. The four rounding modes are:

RNE - Round to nearest or even. Choose the value in the destination precision nearest to the unrounded result. If the unrounded result is midway between two such values, choose the value with its least significant fraction bit as 0 (*even*).

RD - Round down, toward minus infinity.

RU - Round up, toward plus infinity.

RZ - Round toward zero.

The rounding mode is specified by the Rounding Mode field in the Control Register. It is initialized by Thread Dispatch. The normal default value is round to nearest or even. The Rounding Mode can be read to check the mode and written to change it. The Control Register and the Rounding Mode value are thread-specific; the Rounding Mode applies to all floating-point types, all execution channels, and all floating-point instructions executed by the thread after it is assigned.

Rounding an inexact result signals the Inexact Exception.

Rounding an inexact result preserves the sign of the result.

Infinities and NaNs are exact results and are not affected by the rounding mode.

Zeros are exact results, but the signs of zero results are affected by the rounding mode in certain cases:

X - X = +0 for RNE, RU, RZ

X - X = -0 for RD

(+0) + (-0) = (-0) + (+0) = +0 for RNE, RU, RZ

(+0) + (-0) = (-0) + (+0) = -0 for RD

Regardless of the rounding mode, (+0) + (+0) = +0 and (-0) + (-0) = -0.

The *directed rounding modes* are round down, round up, and round toward zero.

In IEEE mode, when a floating-point overflow occurs the result is determined by the sign of the result and the rounding mode:

+ and (RNE or RU): + inf

+ and (RD or RZ): + fmax

- and (RU or RZ): - fmax

- and (RNE or RD): - inf

Note that for floating-point overflow in IEEE mode, RNE always produces ± inf and RZ always produces ± fmax.

Floating-Point Operations and Precision

IEEE Standard 754-2008 requires the following floating-point operations to be precise within <= 0.5 *ulp* (unit in the last place) when using the round to nearest or even rounding mode:

ADD (*add* instruction)

DIV (*math* instruction with FDIV function code)

FMA (fused multiply add, *mad* instruction)

MUL (*mul* instruction)

SQRT (*math* instruction with SQRT function code)

SUB (*add* instruction using one **-** source modifier)

Conversions (float to float, float to int, and int to float)

Min/Max

Compare

Single Precision Floating-Point Rounding to Integral Values

The *rndd* (Round Down), *rnde* (Round to Nearest or Even), *rndu* (Round Up), and the *rndz* (Round to Zero) instructions round arbitary Float values to integral Float values. Each instruction specifies its rounding mode so these instructions are not affected by the Rounding Mode in the Control Register.

An integral source value produces the same value for the destination (ignoring any saturation). For magnitudes >= 8,388,608 (2^{23}) all Float values are integral.

The rounding instructions are sign preserving.

Signed zeros are propagated. In IEEE mode, signed infinities are propagated. In IEEE mode, sNaN inputs are quietized, the equivalent qNaN is produced, and the Invalid Operation exception is indicated. In IEEE mode, qNaN inputs are propagated.

No other exceptions are signaled for these instructions. For example if the source and result values differ, the Inexact exception is not signaled.

The Single Precision Denorm Mode in the Control Register affects the results of the *rndd* and *rndu* instructions for denorm source values.

Floating-Point to Integer Conversion

The *mov* and *sel* instructions can be used to convert floating-point values to integers. In the tables below, Imin is the smallest representable value in a signed integer type, Imax is the largest representable value

in an integer type, and f is a finite floating-point value after rounding to an integral value using the current rounding mode.

Converting unrepresentable floating-point values, including infinities, NaNs, and values that convert to integers outside of the destination type's range, signal Invalid Operation exceptions. When the destination integer type is unsigned, normalized nonzero negative inputs signal Invalid Operation exceptions and negative denorm inputs signal Inexact exceptions.

Data written in accumulator using implicit or explicit destination will not be IEEE compliant.

Floating-Point to Integer Conversion Results and Exceptions for Signed Integer Types

FP Value	Integer Result	FP Exception
qNaN	0	Invalid Operation
sNaN	0	Invalid Operation
+inf	Imax	Invalid Operation
f > Imax	Imax	Invalid Operation
Imin <= f <= Imax	f	Inexact if rounding changed f
f < Imin	Imin	Invalid Operation
-inf	Imin	Invalid Operation

Floating-Point to Integer Conversion Results and Exceptions for Unsigned Integer Types

FP Value	Integer Result	FP Exception
qNaN	0	Invalid Operation
sNaN	0	Invalid Operation
+inf	Imax	Invalid Operation
f > Imax	Imax	Invalid Operation
0 <= f <= Imax	f	Inexact if rounding changed f
f = -0	0	Inexact
-1 < f < -0	0	Inexact
-1 < f < -0	Real Indefinite	Invalid Operation
-fmax <= f <= -1	Real Indefinite	Invalid Operation
-inf	Real Indefinite	Invalid Operation

Note: Real Indefinite is encoded as Integer value 0.

Integer to Floating-Point Conversion

Integer to floating-point conversion follows these rules in IEEE mode:

- The result is never ± inf, never NaN, and never -0.

- If the integer source value is not exactly representable in the destination floating-point format, use the current rounding mode to choose an adjacent floating-point value and signal the Inexact Exception.
- If the integer source value is too large to represent in the destination floating-point format (only possible when converting to Half Float from D, UD, or UW) then signal the Overflow Exception. Based on the sign and the current rounding mode, the result is ± fmax or ± inf, as described in the Overflow Exception section.

Floating-Point Min/Max Operations

In the following Min/Max operations, sNaN inputs are preferred to non-NaN inputs and non-NaN inputs are preferred to qNaN inputs.

Min(x, qNaN) = Min(qNaN, x) = x with no exceptions signaled.

Min(x, sNaN) = Min(sNaN, x) = qNaN (quietized value corresponding to the input sNaN) and signal the Invalid Operation exception.

Note: DX deviates from this rule:

The DX behavior is inferred from the denorm bit.

Min(x, sNaN) = Min(sNaN, x) = x

Max(x, qNaN) = Max(qNaN, x) = x with no exceptions signaled.

Max(x, sNaN) = Max(sNaN, x) = qNaN (quietized value corresponding to the input sNaN) and signal the Invalid Operation exception.

Note: DX deviates from this rule:

The DX behavior is inferred from the denorm bit.

Max(x, sNaN) = Max(sNaN, x) = x

Special cases(when both sources are NaN inputs)

Min(qNaN, qNan) = qNaN (of the first source) and no exception raised

Min(qNaN, sNaN) = Min(sNaN, qNaN) = qNaN (quiet-ized sNaN and signal the Invalid Operation exception)

Min(sNaN, sNaN) = qNaN(of the first source and signal Invalid Operation Exception)

Max(qNaN, qNan) = qNaN (of the first source) and no exception raised

Max(qNaN, sNaN) = Max(sNaN, qNaN) = qNaN (quiet-ized sNaN and signal the Invalid Operation exception)

Max(sNaN, sNaN) = qNaN(of the first source and signal Invalid Operation Exception)

IEEE Floating-Point Exceptions

The EU detects the five floating-point exceptions defined by IEEE Standard 754:

Invalid Operation

Division by Zero

Overflow

Underflow

Inexact

Signaling Floating-Point Exceptions

When enabled in the Control Register, floating-point exceptions are detected and set *sticky* flag bits in the State Register. There is no mechanism for automatic transfer to a handler, so floating-point exceptions are not handled like other exceptions described in the Exceptions chapter.

Setting flags to indicating Floating-Point exceptions is the default exception handling approach specified by IEEE Standard 754. The flag bits are sticky because in normal operation the EU only sets these bits as exceptions occur, and does not clear these bits, so a set value sticks until cleared by software. The Control Register and State Register are cleared at reset and initialized at thread load. Both are read/write registers. These fields are used:

- **IEEE Exception Trap Enable** (Control Register cr0.0:ud bit 9). This bit enables trapping IEEE exception flags. This control bit may be updated by software. It is initially zero on thread load. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exception field of sr0.1, in the State Register. **Note:** Software must set this flag at thread start to use the IEEE Exception flags.
- **IEEE Exception**. (State Register sr0.1:ud bits 7:0). The IEEE exception bits are sticky bits set by the opcodes when floating-point exceptions are triggered. These bits are defined per thread and all channels update one set of sticky bits. These bits are cleared on thread load and may be cleared by software. Exception updates to these bits may be disabled by clearing the IEEE Exception Trap Enable bit in the Control Register. The following table specifies the IEEE exception bits:

Bits	Definition
7:5	Reserved
4	Inexact Exception
3	Overflow Exception
2	Underflow Exception
1	Division by Zero Exception
0	Invalid Operation Exception

The IEEE exception flags are per thread, shared by all channels. (Maintaining separate per channel exception flags for 32 channels would require 160 bits per thread.)

Invalid Operation Exception

An Invalid Operation exception is signaled by any operation on a signaling NaN (sNaN) or by certain combinations of operations and operands with undefined results, always producing a quiet NaN (qNaN) result.

The following specific operations signal Invalid Operation, where *x* is a positive, finite, nonzero, and normalized number:

+inf - (+inf) or (-inf) - (-inf)

± 0 / ± 0

± inf / ± inf

± 0 × ± inf or ± inf × ± 0

Remainder(± *x*, ± 0)

Remainder(± inf, ± *x*)

Sqrt(-*x*)

Note that Sqrt(-0) is -0 per IEEE Standard 754 and does not cause any exception.

These instructions can signal specific Invalid Operation exceptions (and also on sNaN inputs except for Float inputs in ALT mode), producing a qNaN result:

add

dp2, *dp3*, *dp4*, and *dph* (the Dot Product instructions)

line

lrp

mac

mad

madm

math with the FDIV, SQRT, or RSQRTM function codes

mul

pln

These other instructions signal Invalid Operation exceptions on sNaN inputs (except for Float inputs in ALT mode), producing a qNaN result:

cmp, *cmpn*

frc

math with all other function codes for floating-point operations

mov except for raw moves

rndd, *rnde*, *rndu*, and *rndz* (the Round instructions)

sel

smov except for raw moves

Division by Zero Exception

The operation ± *x* / ± 0, where *x* is a positive, finite, nonzero, and normalized number, signals the Division by Zero Exception and produces a correctly signed ± inf result. Note that in accordance with the standard, ± 0 / ± 0 signals Invalid Operation, does not signal Division by Zero, and produces a qNaN result. This behavior can occur for the *math* instruction with the FDIV function code.

The operation $LOG_2(\pm 0)$ signals the Division by Zero Exception and produces -inf as the result. This behavior can occur for the *math* instruction with the LOG function code.

Overflow Exception

A floating-point *overflow* occurs when an operation with a finite result produces an internal result with magnitude > fmax, the maximum representable finite value in the destination format. The internal result is rounded to the destination precision with the current rounding mode but has an unbounded exponent. An overflow produces ± inf or ± fmax as the result, depending on the sign and the rounding mode.

The following algorithm describes floating-point overflow processing:

1. Compute the result with infinite precision and unbounded range.
2. Normalize the result using an unbounded exponent.
3. Round the result to the destination precision using an unbounded exponent and the current rounding mode.
4. If (abs(rounded unbounded result) > fmax(destination format)) {

   Set the Overflow Exception flag to 1.

   Output = ± inf or ± fmax depending on the sign and the rounding mode:

   + and (RNE or RU): + inf

   + and (RD or RZ): + fmax

   - and (RU or RZ): - fmax

   - and (RNE or RD): - inf

   }

   }

Note: The first three steps of the overflow and underflow algorithms are identical.

Underflow Exception

An *underflow* occurs when an operation produces a tiny but nonzero inexact result *x*, with abs(*x*) < fmin, where fmin = 2^{Emin}. See the Floating-Point Type Parameters table for the fmin and Emin values for different floating-point types.

IEEE Standard 754 allows underflow to be determined before or after rounding. The Execution Unit determines underflow after rounding, which is consistent with the behavior of the x87 and SSE instructions in the CPU.

When denorms are flushed to zero, no underflow exceptions are signaled. Flush to zero is not defined by IEEE Standard 754, but is implementation specific and required by some APIs (including DirectX), thus the EU ISA for BDW supports either using or flushing Float or Double Float denorms based on the respective Denorm Mode bits.

When denorms are enabled, if an operation's internal result rounded to the destination precision, but using an unbounded exponent range, has a magnitude that is less than fmin but nonzero AND the denorm result in the destination format is inexact, then signal the Underflow Exception.

The rounded result can be ± 0, ± fmin, or (when denorms are enabled) ± denorm.

IEEE Standard 754 requires the non-intuitive behavior that an exact denorm result does not set the Underflow Exception flag.

The following algorithm describes floating-point underflow processing:

1. Compute the result with infinite precision and unbounded range.
2. Normalize the result using an unbounded exponent.
3. Round the result to the destination precision using an unbounded exponent and the current rounding mode.
4. If (0 < abs(rounded unbounded result) < fmin(destination format) {

   if (flush denorms to zero) {

   Output = 0; // No underflow exception.

   }

   Else {

   Renormalize to the bounded exponent with the original infinite precision value...

   ...and round that value to the destination precision using the current rounding mode.

   If (the just computed value differs from the value computed in step (3) in exponent or mantissa) {

   Set the Underflow Exception flag to 1.

   } // Note: Underflow is not set if the tiny result is the same as when computed with an unbounded exponent.

   Output = rounded result using the destination precision and destination exponent range;

}

}

Else { // Not a tiny number.

Output = rounded number;

}

Inexact Exception

An Inexact Exception occurs when the internal unrounded result, with infinite precision and unbounded exponent range, differs from the generated result after format conversion, normalizing or denormalizing, and rounding. An Inexact Exception occurs irrespective of any saturation to exact zero or exact +1.0. The Inexact Exception is normal and may occur more often than not. For example, the calculation 1.0 / 3.0 is inexact in any binary floating-point format. These rules determine whether a result is inexact:

- Infinities and NaNs are never inexact.
- Flushing a denorm internal result to zero (always for Half Float and if the appropriate Denorm Mode is 0 for Float and Double Float) is always inexact.
- If any rounding occurs using the current Rounding Mode, so the rounded result differs from the internal unrounded result, the result is inexact. However explicit round to integral using any of the rounding instructions (*rndd*, *rnde*, *rndu*, and *rndz*) is never inexact.

Floating-Point Compare Operations

Four mutually exclusive relations are possible between two floating-point values, src0 and src1:

Less than. src0 < src1 and neither source is NaN.

Equal. src0 = src1 and neither source is NaN.

Greater than. src0 > src1 and neither source is NaN.

Unordered. Any source is NaN.

Any NaN compares unordered to any value, including itself.

Infinities of the same sign compare as equal.

Zeros compare as equal regardless of sign: -0 = +0.

Floating-Point Compare Relations

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	E	L	L	L	L	L	L	L	U
-fin	G	*	L	L	L	L	L	L	U
-denorm	G	G	*^	L^	L^	L^	L	L	U
-0	G	G	G^	E	E	L^	L	L	U
+0	G	G	G^	E	E	L^	L	L	U
+denorm	G	G	G^	G^	G^	*^	L	L	U
+fin	G	G	G	G	G	G	*	L	U
+inf	G	G	G	G	G	G	G	E	U
NaN	U	U	U	U	U	U	U	U	U

Notes	
*	Relation can be L, E, or G.
^	When denorms are flushed to zero then all denorms and zeros compare as E.

The next six tables show the results of six specific comparisons, corresponding to the **.g**, **.l**, **.e**, **.ne**, **.ge**, and **.le** conditional modifiers used with the *cmp* instruction and a floating-point source type. Any NaN source produces a false comparison result for these modifiers other than **.ne** and produces a true comparison result for the **.ne** modifier.

Results of Greater Than Comparison — *cmp*.g

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	F	F	F^	F	F	F
+0	T	T	T^	F	F	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.g**

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	F	F	F^	F	F	F
+0	T	T	T^	F	F	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
	comparison result F.

Results of Less Than Comparison — *cmp*.l

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	F
-fin	F	*	T	T	T	T	T	T	F
-denorm	F	F	*^	T^	T^	T^	T	T	F
-0	F	F	F^	F	F	T^	T	T	F
+0	F	F	F^	F	F	T^	T	T	F
+denorm	F	F	F^	F^	F^	*^	T	T	F
+fin	F	F	F	F	F	F	*	T	F
+inf	F	F	F	F	F	F	F	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.l** comparison result F.

Results of Equal Comparison — *cmp*.e

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	F
-fin	F	*	F	F	F	F	F	F	F
-denorm	F	F	*^	F^	F^	F^	F	F	F
-0	F	F	F^	T	T	F^	F	F	F
+0	F	F	F^	T	T	F^	F	F	F
+denorm	F	F	F^	F^	F^	*^	F	F	F
+fin	F	F	F	F	F	F	*	F	F
+inf	F	F	F	F	F	F	F	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.e** comparison result T.

Results of Not Equal Comparison — *cmp*.ne

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	T
-fin	T	*	T	T	T	T	T	T	T
-denorm	T	T	*^	T^	T^	T^	T	T	T
-0	T	T	T^	F	F	T^	T	T	T
+0	T	T	T^	F	F	T^	T	T	T
+denorm	T	T	T^	T^	T^	*^	T	T	T
+fin	T	T	T	T	T	T	*	T	T
+inf	T	T	T	T	T	T	T	F	T
NaN	T	T	T	T	T	T	T	T	T

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.ne** comparison result F.

Results of Greater Than or Equal Comparison — *cmp*.ge

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	T	T	F^	F	F	F
+0	T	T	T^	T	T	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes

*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.ge** comparison result T.

Results of Less Than or Equal Comparison — *cmp*.le

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	T	T	T	T	T	T	T	F
-fin	F	*	T	T	T	T	T	T	F
-denorm	F	F	*^	T^	T^	T^	T	T	F
-0	F	F	F^	T	T	T^	T	T	F
+0	F	F	F^	T	T	T^	T	T	F
+denorm	F	F	F^	F^	F^	*^	T	T	F
+fin	F	F	F	F	F	F	*	T	F
+inf	F	F	F	F	F	F	F	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes

*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.le** comparison result T.

Type Conversion

This topic is currently under development.

Float to Integer

Converting from float to integer is based on rounding toward zero(RTZ is for DX, IEEE expects all four rounding modes). If the floating point value is +0, -0, +Denorm, -Denorm, +NaN –r -NaN, the resulting integer value is always 0. If the floating point value is positive infinity (or negative infinity), the conversion result takes the largest (or the smallest) represent-able integer value. If the floating point value is larger (or smaller) than the largest (or the smallest) represent-able integer value, the conversion result takes the largest (or the smallest) represent-able integer value. The following table shows these special cases. The last two rows are just examples. They can be any number outside the represent-able range of the output integer type (UD, D, UW, W, UB and B).

Input Format	Output Format					
F	UD	D	UW	W	UB	B
+/- Zero	00000000	00000000	00000000	00000000	00000000	00000000
+/- Denorm	00000000	00000000	00000000	00000000	00000000	00000000
NAN	00000000	00000000	00000000	00000000	00000000	00000000
-NAN	00000000	00000000	00000000	00000000	00000000	00000000
INF	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F
-INF	00000000	80000000	00000000	00008000	00000000	00000080
$+2^{32}$ (*)	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F
$-2^{32}-1$ (*)	00000000	80000000	00000000	00008000	00000000	00000080

Integer to Integer with Same or Higher Precision

Converting an unsigned integer to a signed or an unsigned integer with higher precision is based on zero extension.

Converting an unsigned integer to a signed integer with the same precision is based on modular wrap-around. Without saturation, a larger than represent-able number becomes a negative number. With saturation, a larger than represent-able number is saturated to the largest positive represent-able number.

Converting a signed integer to a signed integer with higher precision is based on sign extension.

Converting a signed integer to an unsigned integer with higher precision is based on sign extension. Without saturation, a negative number becomes a large positive number with the sign bit wrapped-up. With saturation, a negative number is saturated to zero.

Integer to Integer with Lower Precision

Converting a signed or an unsigned integer to a signed or an unsigned integer with lower precision is based on bit truncation. Without saturation, only the lower bits are kept in the output regardless of the sign-ness of input and output. With saturation, a number that is outside the represent-able range is saturated to the closest represent-able value.

Integer to Float

Converting a signed or an unsigned integer to a single precision float number is to round to the closest representable float number. For any integer number with magnitude less than or equal to 24 bits, resulting float number is a precise representation of the input. However, if it is more than 24 bits, by default a “round to nearest even” is performed.

Double Precision Float to Single Precision Float

.

Double Precision Float	Single Precision Float
-inf	-inf
-finite	-finite/-denorm/-0
-denorm	-0
-0	-0
+0	+0
+denorm	+0
+finite	+finite/+denorm/+0
+inf	+inf
NaN	NaN

The upper Dword of every Qword will be written with undefined value when converting DF to F.

Single Precision Float to Double Precision Float

Converting a single precision floating-point number to a double precision floating-point number will produce a precise representation of the input.

Single Precision Float	Double Precision Float

Single Precision Float	Double Precision Float
-inf	-inf
-finite	-finite
-denorm	-finite
-0	-0
+0	+0
+denorm	+finite
+finite	+finite
+inf	+inf
NaN	NaN

Exceptions

The GEN Architecture defines a basic exception handling mechanism for several exception cases. This mechanism supports both normal operations such as extensions of the mask-stack depth, as well as detecting some illegal conditions.

Exception Types

Type	Trigger / Source	Sync/Async Recognition
Software Exception	Thread code	Synchronous
Breakpoint	• A bit in the instruction word • Breakpoint IP match • Breakpoint Opcode match	Synchronous
Illegal Opcode	Hardware	Synchronous
Halt	MMIO register write	Asynchronous
Context Save/Restore	Preemption Interrupt	Asynchronous

Threads may choose which exceptions to recognize and which to ignore. This mask information is specified on a per-kernel basis in fixed function state generated by the driver, and delivered to the EU as part of a new thread dispatch. Upon arrival at the EU, the exception-mask information is used to initialize the exception enable fields of that thread's cr0.1 register, which controls exception recognition. This register is instantiated on a per-thread basis, allowing independent control of exception type recognition across hardware threads. The exception enable bits in the cr0.1 register are read/write, and thus can be enabled/disabled via software at any time during thread execution.

The exception handling mechanism relies on the System Routine, a single subroutine that provides common exception handling for all threads on all EUs in the system. This System Routine is defined per-context and is identified via a System IP (SIP) register in context state. At the time of each context switch,

the appropriate SIP for that context is loaded into each EU, allowing each context to have custom implementation of exception handling routines if so desired.

The mechanism does not support handling recursive system routine access. This means when a thread cannot be asynchronously interrupted to an exception when executing a SIP.

Example:

An Exception is not supported when hardware is executing a SIP for context save and restore operations.

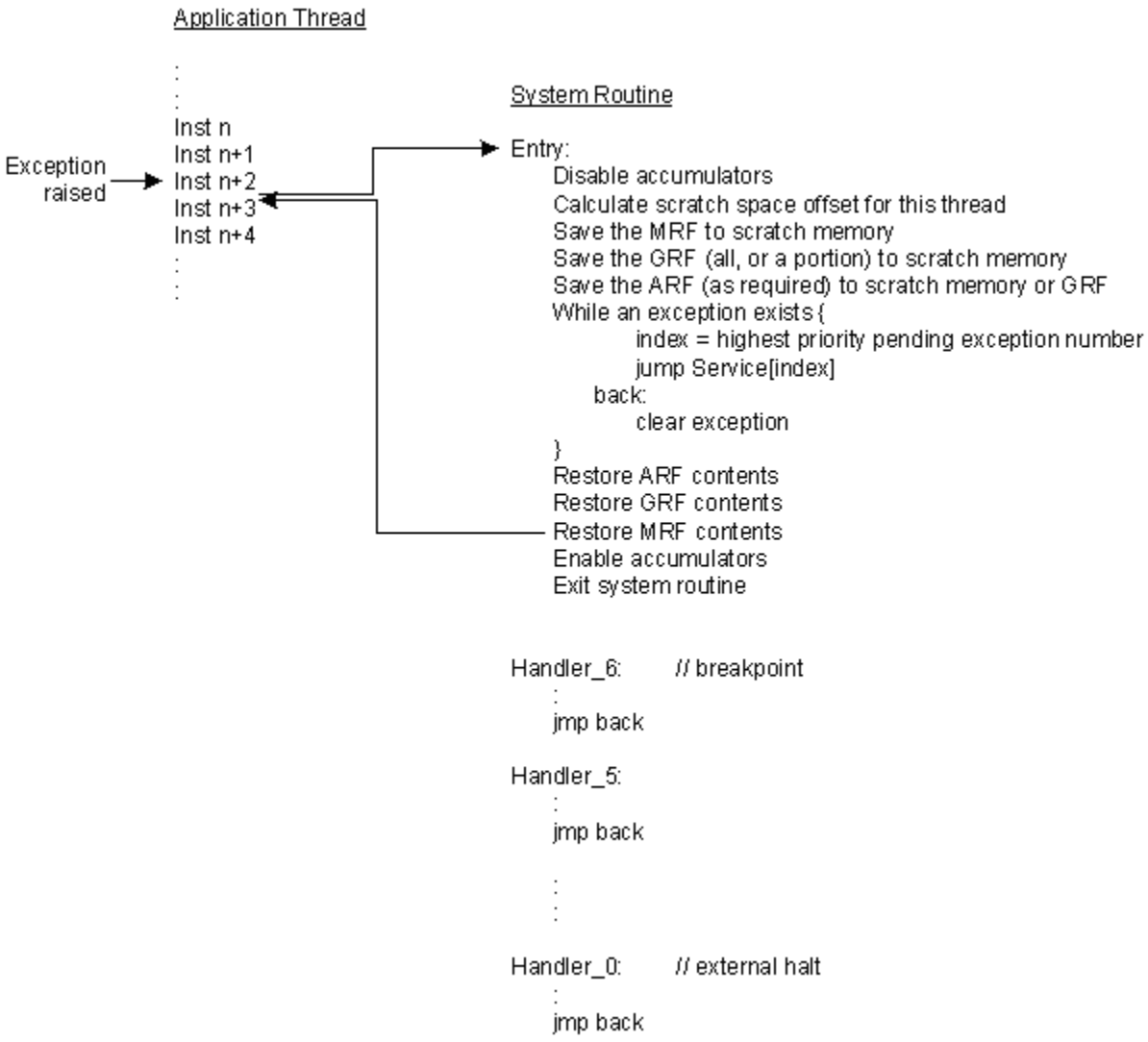
Exception-Related Architecture Registers

Exception-related registers are architecture registers cr0.0 through cr0.2. These registers are instantiated on a per-thread basis providing each hardware thread with unique control over exception recognition and handling. The registers provide the capability to mask exception types, determine the type of raised exception, store the return address, and control exiting from the System Routine back to the application thread.

Many of the bits in these registers are manipulated by both hardware and software. In all cases, the read/write operations by hardware and software occur at exclusive times in a thread's lifetime, thus there is no need for atomic read-modify-write operations when accessing these registers.

System Routine

The following diagram illustrates the basic flow of exception handling and the structure of the System Routine.



Invoking the System Routine

The System Routine is invoked in response to a raised exception. Once an exception is raised, no further instructions from the application thread are issued until the System Routine has executed and returned control back to the application thread.

After an exception is recognized by hardware, the EU saves the thread's IP into its AIP register (cr0.2), and then moves the System Routine offset, SIP, into the thread's IP register. At this point the next instruction issued for that thread is the first instruction of the System Routine.

The System Routine maintains the same execution priority, GRF register space, and thread state as the application thread from which it is invoked.

Due to assuming the same priority, there may be significant absolute time between an exception being raised and invoking the System Routine, as other higher priority threads within the EU continue to execute. From a thread's perspective, once an exception is recognized, the next instruction issued is from the System Routine.

At the time of System Routine invocation, there may still be outstanding registers in-flight from the application thread. Depending on the instruction sequence in the System Routine, an in-flight register may be referenced by the System Routine and cause a register-in-flight dependency. These dependencies are honored by the System Routine and may cause the System Routine to be suspended until the register retires.

Exception processing is not nested within the System Routine. If a future exception is detected while executing the System Routine, the exception is latched into cr0.1, but does not cause a nested re-invocation of the System Routine. The exception recognition hardware recognizes only one outstanding exception of each type; i.e., once a specific exception type is detected and latched in cr0.1, and until the exception is cleared, any further exception of that type is lost.

Accumulators are not natively preserved across the System Routine. To make sure the accumulators are in the identical state once control is returned to the application thread, the System Routine must either set the Accumulator Disable bit of cr0.0 before using any instruction that modifies an accumulator, or save and restore the accumulators (using GRF registers or system thread scratch memory) around the System Routine. Saving and restoring accumulators, including their extended precision bits, can be accomplished by a short series of moves and shifts of the accumulator register. Also note that the state of the Accumulator Disable bit itself must be preserved unless, by convention, the driver software limits its manipulation to only the System Routine.

Further, upon System Routine entry, the execution-related masks (Continue, Loop, If, and Active masks, contained in the Mask Register) will remain set as they were in the application thread. Thus only a subset of channels may be active for execution. To enable execution on all channels, the System Routine may choose to use the instruction option ‘NoMask’, or may choose to set the mask registers to the desired value so long as it saves/restores the original masks upon System Routine entry/exit.

Similarly there is no hardware mechanism to preserve flags, mask-stacks, or other architecture registers across the System Routine. The System Routine must ensure that these values are preserved (see the *Conditional Instructions Within the System Routine*section for related information).

Returning to the Application Thread

Prior to returning control to the application thread, the System Routine should clear the proper Exception Status and Control bit in cr0.1. Failure to do so forces the thread's execution to reenter the System Routine before any further instructions are executed from the application thread. (Note that single-stepping functionality is the one exception where the exception's Status and Control bit is not reset before exit.)

The System Routine may choose to loop within a single invocation of the System Routine until all pending exceptions are serviced, or may choose to service exceptions one at a time (a simpler solution, but less efficient).

The System Routine is exited, and control returned to the application thread, via a write to the Master Exception State and Control bit in cr0.0. Upon clearing this bit, the value of AIP (cr0.2) is restored to the thread's IP register and, with no further exceptions pending, execution resumes at that address. The System Routine must follow any write to the Master Exception State and Control bit with at least one SIMD-16 *nop* instruction to allow control to transition. Throughout the System Routine, the AIP register maintains its value at the time the exception was raised unless directly modified by the System Routine. (See the AIP register definition for specifics on the IP value saved to AIP).

System IP (SIP)

The System IP (SIP) is the 16 byte-aligned offset of the first instruction of the System Routine, relative to the General State Base Address. SIP is assigned by the STATE_SIP command to the command streamer which updates SIP in the EU.

The SIP is widened to 48 bits. However the EU still only uses the low 32 bits (bits 31:4 with bits 3:0 as zero).

When the System Routine is invoked, the application thread's current IP is first saved into the AIP field of cr0.2. The SIP address is then loaded into the thread's IP register and execution continues within the System Routine. Thus each invocation of the System Routine has a common entry point. Returning from the System Routine loads IP from AIP, continuing thread execution.

System Routine Register Space

The System Routine uses the same GRF space as the thread that invokes it.

As such all of the calling thread's registers and their contents are visible to the System Routine. Further, the System Routine must only use r0..r15 of the GRF, as a minimal thread may have requested and been allocated this few. If the System Routine requires more registers than this, the driver should establish a higher minimum allocation for all threads.

The System Routine may encounter any residual register dependencies of the calling thread until such time that they clear by the return of in-flight writebacks.

Only one 32-bit GRF location, r0.4, is reserved for System Routine use. This location is sufficient to allow the System Routine to calculate the appropriate offset of its private scratch memory in the larger system scratch memory space (as dictated by binding table entry 254). The offset is left as a driver convention, but is likely based on a combination of Thread and EU IDs (see the example system handler in the *System Scratch Memory Space*) section. Other than the reserved r0.4 register field, there is no explicit GRF register space dedicated to the System Routine, and any GRF needs must be accomplished via (a) convention between the System Routine and application code, or (b) the System Routine temporarily spilling the thread's GRF register contents to scratch memory and restoring those contents before System Routine exit.

No persistent storage is automatically allocated to the System Routine, although a driver implementation may set aside part of system scratch memory for the System Routine.

Any parameter passing to the System Routine (for use by software exceptions) is done via the GRF based on system thread/application thread convention.

System Scratch Memory Space

There is a single unified system scratch memory space per context shared by all EUs. It is anticipated that block is further partitioned into a unique scratch sub-space per thread via conventions implemented in the System Routine, with each hardware thread having a uniform block size at a calculated offset from the base address. The block address for a thread can be based on an offset derived from the thread's execution unit ID and thread ID made available through the TID and EUID field of architecture register sr0.0.

Per_Thread_Block_Size = System_Scratch_Block_Size / (EU_Count * Thread_Per_EU);

Offset = (sr0.0.EID * Threads_Per_EU + sr0.0.TID) * Per_Thread_Block_Size;

where in GEN:

Threads_Per_EU = 4

EU_Count = 8

System_Scratch_Block_Size is a driver choice.

Access to system scratch memory is performed through the Data Port via linear single register or block-based read/write messages. The driver may choose to use any binding table index for system scratch surface description. As a practical matter, the same index is expected to be used across all binding tables, as the index is typically hard coded in Data Port messages used within the System Routine coupled with the fact that a single System Routine is used for all threads. Read/write messages to the Data Port contain the address of the binding table (provided in r0 of all threads) and an offset, from which the Data Port calculates the final target address.

It is expected that the system scratch memory space is allocated by the driver at context-create time and remains persistent at a constant memory address throughout a context's lifetime.

Conditional Instructions Within the System Routine

It is expected that most, if not all, control flow within the System Routine is scalar in nature. If so, the System Routine should set SPF (Single Program Flow, cr0.0) to enable scalar branching. In this mode, conditional/loop instructions do not update the mask stacks and therefore do not have restrictions on their use nor require the save/restore of hardware mask stack registers.

If SIMD branching is desired within the System Routine, special considerations must be taken. Upon entry to the System Routine, the depth of the mask stacks is unknown at that point, and may be near full. If so, a subsequent conditional instruction and its associated mask ‘push’ may cause a stack overflow. This would generate an exception within the system routine, an unsupported occurrence. To prevent this, if the System Routine uses SIMD conditional instructions, it must save the mask stacks prior to the first

SIMD conditional instruction, and restore them after the last SIMD conditional instruction. As a general solution, it may be easiest to simply implement the save/restore as part of the entry/exit code sequence, using an available GRF register pair as a storage location. Once saved, the stacks should be reset to their empty condition, namely depth = 0 and top of stack = 0xFFFFFFFF.

Use of NoDDClr

The GEN instruction word defines an instruction option **NoDDClr** that overrides the native register dependency clearing mechanism of the typical instruction. When specified, NoDDClr does not clear, at register writeback time, the dependency placed on the destination register of the instruction. Use of this mechanism may provided increased performance when a kernel can guarantee no dependency issues between instructions, but may cause issues with exception handling in some circumstances as discussed here.

Typically NoDDClr is used in an instruction series to enable a sequence of writes to sub-fields of a GRF register without paying a dependency penalty on each instruction. In this case, NoDDClr and NoDDChk are used across an instruction sequence to allow the first instruction to set the destination dependency, interior instructions to write to the GRF register without dependency checks, and the last instruction to clear the dependency. (This sequence is referred to as a NoDDClr code block going forward). By only allowing the last instruction to clear the dependency, program execution is prevented from going beyond a certain point until all writes of that sequence are known to retire.

The problem arises if an exception is raised within a NoDDClr code block. In this case, there exists the potential for the System Routine to hang while attempting to save/restore a register used as a destination register by the NoDDCLr code block, as the outstanding dependency on that register will not clear until the final instruction of the NoDDCLr block is executed, sometime after the System Routine returns. Should the System Routine attempt to use that register, it hangs waiting on a dependency to be cleared by an instruction not yet issued.

Note: This is a known condition and will in some cases not allow the full GRF contents to be externally visible in System Routine scratch space during a break or halt exception.

To avoid this condition, guidelines are provided below for consideration. (Note that these are general guidelines, some of which can be alleviated through careful coding and register usage conventions and restrictions.)

Guideline
NoDDClr code blocks should only be used where absolutely necessary.
Instructions that may generate exceptions should not be placed within NoDDClr blocks. This includes most conditional branch instructions (if, do, while, ...).
If possible, use NoDDClr on registers high in the thread's register allocation (e.g. r120), thus even if a System Routine hang occurs, as much of the GRF is visible as possible. (Note that this would also require the System Routine to update the progress of the GRF dump, perhaps with each GRF block written, or to initialize the System Routine's scratch space to a known value, to be able to distinguish valid/locations from unwritten locations).

Also a driver implementation may consider a disable-NoDDclr option in which jitted code does not use the NoDDClr capability. In this case, there is no change to the code that is jitted other than removal of

the NoDDClr instruction option. The code executes as normal, but with a higher number of thread switches in what would have been a NoDDClr code block.

Exception Descriptions

This section describes conditions that can cause exceptions and transfer control to the System Routine.

Illegal Opcode

The GEN ISA defines a single *illegal* opcode. The byte value of the *illegal* opcode is 0x00 due to it being a likely byte value encountered by a wayward instruction pointer value. The *illegal* instruction signals an exception if exception handling is enabled and invokes the system interrupt routine. If exception handling is NOT enabled, the illegal opcode is executed resulting in undetermined behavior including a system hang. Hardware decodes all legal opcodes supported. Any byte value that is not in the legal opcode list is decoded as an illegal opcode to trigger exception.

Undefined Opcodes

All undefined opcodes in the 8-bit opcode space (which includes instruction bit 7, reserved for future opcode expansion) are detected by hardware. If an undefined opcode is detected, the opcode is overridden by hardware, forcing the opcode value within the pipeline to the defined *illegal* opcode. The offending instruction, should it eventually be issued down the execution unit's pipeline, generates an Illegal Opcode exception as described in the section *Illegal Opcode*. The memory location of the offending opcode keeps its original value. That location can be queried to determine the opcode value.

Software Exception

A mechanism is provided to allow an application thread to invoke an exception and is triggered using the Software Exception Set and Clear bit of cr0.1. Sub-function determination and parameter passing into and out of the exception handler is left to convention between the system-thread and application-thread. The thread's IP is incremented before saving AIP and entering the System Routine, causing execution to resume at the next application-thread instruction after returning from the System Routine.

Context Save and Restore

The System Routine is also used to save and restore the context of the Execution Unit. This feature is enabled in GPGPU workloads *only*.

When the execution engine receives a preemption or an interrupt, the application thread invokes the System Routine. The System Routine is invoked only when all in-flight registers have retired. The system routine is used to save all the state of the EU to memory. When the sequence is complete, the master exception control bit is cleared. This action stops all execution for the given thread and invalidates the thread. This means a new thread from a different context may be loaded. When the master exception control bit is cleared, software must ensure that all outstanding messages from the EU are dispatched out of the execution message pipeline. This is achieved by creating a dependency on the last send that is

saving EU state. A dummy instruction before clearing the master exception control bit ensures that this is achieved.

The System Routine is also invoked on a context restore request. In this case a dummy thread is loaded into the EU which starts with the System Routine. This routine now restores the state of the EU. The restore sequence used in such a case should be consistent with the save sequence to ensure that state is restored correctly. After completing the restore sequence, the System Routine must clear the master exception control bit in the Control Register. This enables hardware to switch to the application thread which continues execution.

Programming Note	
Context:	Context Save and Restore
When context save and restore is required to be supported for GPGPU work loads, Stack Overflow exception handling will not be supported. Software will either need to ensure stack is either completely disabled OR used in such a way, an exception will not trigger.	

Events That Do Not Generate Exceptions

The conditions described in this section are either not recognized or do not generate an exception.

Illegal Instruction Format

This condition includes malformed instructions in which the opcode is legal, but the source or destination operands or other instruction attributes do not comply with the instruction specification. There is no direct hardware support to detect these cases and the outcome of issuing a malformed instruction is undefined.

Note that GEN does not support self-modifying code, therefore the driver has an opportunity to detect such cases before the thread is placed in service.

Malformed Message

A message's contents, destination registers, lengths, and descriptors are not interpreted in any way by the execution unit. Errors in specifying message fields do not raise exceptions in the EU but may be detected and reported by the shared functions.

GRF Register Out of Bounds

Unique GRF storage is allocated to each thread which, at a minimum, satisfies the register requirements specified in the thread's declaration. References to GRF register numbers beyond that called for in the thread's declaration do not generate exceptions. Depending on the implementation, out-of-bounds register numbers may be remapped to r0..r15, although this functionality should not be relied upon by the thread. The hardware guarantees the isolation of each thread's register space, thus there is no possibility of direct register manipulation via an out-of-bounds register access.

Hung Thread

There is no hardware mechanism in the EU to detect a hung thread and such a thread may remain hung indefinitely. It is expected that one or more hung threads will eventually cause the driver to recognize a context timeout and take appropriate recovery action.

Instruction Fetch Out of Bounds

The EU implements a full 32-bit instruction address range (with the 4 LSBs don't care), making it possible for a thread to attempt to jump to any 16-byte aligned offset in the 32-bit instruction address range. (Instruction addresses are offsets from the General State Base Address.) The EU does not provide any type of address checking on instruction fetch requests sent to the memory/cache hierarchy, although error conditions for memory addresses are reported via the Page Table Error Register and other memory interface registers.

FPU Math Errors

The EU's floating point units (FPUs) have defined behaviors for traditional floating point errors and do not generate exceptions. There is no support for signaling FPU math errors as exceptions.

Adds the IEEE Exception Trap Enable bit, which enables trapping IEEE exception flags. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exceptions field of sr0.1. Note that IEEE floating-point exceptions still do *not* transfer control to any handler.

Computational Overflow

Depending on source operand types and values, destination type, and the operation being performed, overflows may occur in the execution pipelines. Many instructions support the overflow (**.o**) conditional modifier that assigns flag bits based on whether or not an overflow occurs.

The EU never signals exceptions for overflows. Software must provide any overflow handling.

System Routine Example

The following code sequence illustrates some concepts of the System Routine. It is intended to be just a shell, without getting into the specifics of each exception handler.

This example contains SNB code for the message registers in the MRF. All message register and MRF references are specific to SNB. Other code in this example is useful for other processor generations.

The example frees enough MRF and GRF space to get the routine started, then jumps to the handler for the specific exception. Many other implementations are also valid, including single exception servicing (as opposed to looping) per invocation, and saving only the GRF or MRF space required by the exception being serviced.

```
#define ACC_DISABLE_MASK 0xFFFFFFFD
#define MASTER_EXCP_MASK 0x7FFFFFFF
#define SYSROUTINE_SCRATCH_BLKSIZE 16384 // for example
```

```
// Shared function IDs:
#define DPR 0x04000000
#define DPW 0x05000000

// Message lengths:
#define ML5 0x00500000
#define ML9 0x00900000

// Response lengths:
#define RL0 0x00000000
#define RL4 0x00040000
#define RL8 0x00080000

// Data port block sizes:
#define BS1_LOW  0x0000
#define BS1_HIGH 0x0100
#define BS2      0x0200
#define BS4      0x0300

// Scratch Layout:
#define SCR_OFFSET_MRF 0           // MRF is specific to SNB.
#define SCR_OFFSET_GRF 512         // + 16 MRF registers
#define SCR_OFFSET_ARF 512 + 4096  // + 16 MRF + 128 GRF registers

// Write data port constants:
// target=dcache, type= oword_block_wr, binding_tbl_offset=0
#define DPW 0x000

// Read data port constants:
// target=dcache, type= oword_block_rd, binding_tbl_offset=0
#define DPR 0x000

Sys_Entry:  // Entry point to the System Routine.

// Disable accumulator for system routine:
and (1) cr0.0 cr0.0 ACC_DISABLE_MASK {NoMask}

// Calc scratch offset for this thread into r0.4:
shr (1) r0.4 sr0.0:uw 6 {NoMask}
add (1) r0.4 r0.4 sr0.0:ub {NoMask}
mul (1) r0.4 r0.4 SYSROUTINE_SCRATCH_BLKSIZE {NoMask}

// Setup m0 with block offset:
mov (8) m0 r0{NoMask}

// Save MRF 7..0 (may choose to save the whole MRF). MRF is specific to SNB:
add (1) m0.2 r0.4 SCR_OFFSET_MRF {NoMask}
send (8) null m0 null DPW|ML9|RL0 {NoMask}

// Save MRF 8..15 (optional; req'ed if sys-routine stays w/in mrf7-0). MRF is specific to
SNB.
mov (8) m7 r0 {NoMask}
add (1) m7.2 r0.4 (SCR_OFFSET_MRF + 256) {NoMask}
send (8) null m7 null DPW|ML9|RL0 {NoMask}

// Save r0..r1 to system scratch:
// Note: done as a single register to guarantee external visibility
// See Use of NoDDClr mov (16) m1 r0 {NoMask}
send (8) m0 null null DPW|ML2|RL0 {NoMask}

// Save r2..r3 to free some room:
mov (16) m3 r2 {NoMask}
```

```
add (1) m0.2 r0.4 SCR_OFFSET_GRF + 64 {NoMask}
send (8) m0 null null DPW|ML4|RL0 {NoMask}

// Save r4..r7 to free some room (optional, depending on needs):
mov (16) m8 r4 {NoMask}
mov (16) m10 r6 {NoMask}
add (1) m7.2 r0.4 (SCR_OFFSET_GRF + 128) {NoMask}
send (8) m7 null null DPW|ML5|RL0 {NoMask}

// Save r8..r11 to free some room (optional, depending on needs):
mov (16) m1 r8 {NoMask}
mov (16) m3 r10 {NoMask}
add (1) m0.2 r0.4 (SCR_OFFSET_GRF + 256) {NoMask}
send (8) m0 null null DPW|ML5|RL0 {NoMask}

// Save r12..r15 to free some room (optional, depending on needs):
mov (16) m8 r12 {NoMask}
mov (16) m10 r14 {NoMask}
add (1) m7.2 r0.4 (SCR_OFFSET_GRF + 384) {NoMask}
send (8) m7 null null DPW|ML5|RL0 {NoMask}

// Save selected ARF registers (optional, depending on use):
// flags, others ...
// Save f0.0:
mov (1) r1.0:uw f0.0 {NoMask}

Next:  // Exceptions pending? If not, exit.

cmp.e (1) f0.0 cr0.4:uw 0:uw {NoMask}
(f0.0) mov (1) IP EXIT {NoMask}

// Find highest priority exception:
lzd (1) r1.1:uw cr0.4:uw {NoMask}

// Jump table to service routine:
jmpi (1) r1.1:uw{NoMask}
mov (1) IP CRService_0 {NoMask}
mov (1) IP CRService_1 {NoMask}
mov (1) IP CRService_2 {NoMask}
...
mov (1) IP CRService_15{NoMask}
mov (1) IP Next

Service_0:
// Clear exception from cr0.1.
// Perform service routine.
// Jump to exit (or if looping on exceptions, go to next loop).
...
Service_15:
// Clear exception from cr0.1.
// Perform service routine.
// Jump to exit (or if looping on exceptions, go to next loop).

Exit:
// Restore f0.0.
// Restore other ARF registers (as required).
// Restore r12..r15.
// Restore r8..r11.
// Restore r4..r7.
// Restore r0..r3.
// Restore m8..m15.
// Restore m0..m7.
```

```
// Clear Master Exception State bit in cr0.0:
and (1) cr0.0 cr0.0 MASTER_EXCP_MASK
nop (16)
```

Below is a code sequence to programmatically clear the GRF scoreboard in case of a timeout waiting on a register that may never return.

At this point, all we know is we have a hung thread. We'd like to copy the GRF to scratch memory to make it visible, but there may be a register that is hung with an outstanding dependency. To get around any hung dependency, walk the GRF using NoDDChk, using an execution mask of f0 == 0 so we don't touch the register contents.

```
Clear_Dep:
mov f0 0x00
(f0) mov r0 0x00 {NoDDChk}
(f0) mov r1 0x00 {NoDDChk}
(f0) mov r2 0x00 {NoDDChk}
...
(f0) mov r127 0x00 {NoDDChk}
// GRF scoreboard now cleared.
```

Instruction Set Summary

Instruction Set Characteristics

SIMD Instructions and SIMD Width

GEN instructions are SIMD (single instruction multiple data) instructions. The number of data elements per instruction, or the execution size, depends on the data type. For example, the execution size for GEN instructions operating on 256-bit wide vectors can be up to 8 for 32-bit data types, and be up to 16 for 16-bit data. The maximum execution size for GEN instructions for 8-bit data types is also limited to 16.

An instruction compression mode is supported for 32-bit instructions (including mixed 32-bit and 16-bit data computation). A compressed GEN instruction works on twice as much SIMD data as that for a non-compressed GEN instruction. A compressed instruction is converted into two native instructions by the instruction dispatcher in the EU.

GEN instructions are executed on a narrower SIMD execution pipeline. Therefore, GEN native instructions take multiple execution cycles to complete.

Instruction Operands and Register Regions

Most GEN instructions may have up to three operands, two sources and one destination. Each operand is able to address a register region. Source operands support negate and absolute modifier and channel swizzle, and the destination operand supports channel mask.

Dual destination instructions are also supported (four-operand instructions in a general sense): One case is for the implied destination – flag register, where the conditional modifiers and the predicate modifiers

may apply. Another case is the message header creation (implied move or implied assembling of the header) in the *send* instruction.

Each execution channel contains an accumulator that is wider than the input data to support back-to-back accumulation operations with increased precision. The added precision (see accumulator register description in Execution Environment chapter) determines the maximum number of accumulations before possible overflow. The accumulator can be pre-loaded through the use of *mov*. It can also be pre-loaded by arithmetic instructions such as *add* or *mul*, since the result of these instructions can go to the accumulator. The accumulator registers are per thread and therefore safe for thread switching.

Register access can be direct or register-indirect. Register-indirect register access uses address registers plus an immediate offset term to compute the register addresses, and only applies to the first source operand (src0) and/or the destination operand.

There is one address register.

Description
There are 16 address sub-registers.

Each sub-register contains a 16-bit unsigned value. The leading two sub-registers form a special doubleword that can be used as the descriptor for the send instruction.

Source operand can also be immediate value (also referred to as inline constants). For instructions with two source operands, only the second operand src1 is allowed to be immediate. For instructions with only one source operand, the source operand src0 is used and it can be an immediate.

An immediate source operand can be a scalar value of specified type up to 32-bit wide, which is replicated to create a vector with length of Execution Size. An immediate operand can also be a special 32-bit vector with 8 elements each of 4-bit signed integer value, or a 32-bit vector with 4 elements each of 8-bit restricted float value.

Instruction Execution

It is implied that all instructions operate across all channels of data unless otherwise specified either via destination mask, predication, execution mask (caused by SIMD branch and loop instructions), or execution size.

Instruction execution size can be specified per instruction, from scalar (*ExecSize* = 1) up to the maximal execution size supported for the data type, with the restriction that execution size can only be in power of 2.

Instruction Formats

This section shows the machine formats of the GEN instruction set. The instructions in the GEN architecture have a fixed length of 128 bits in the native format. A compact format, discussed separately in this volume, can represent some instructions using 64 bits. Out of the 128 bits in the native format, there are 120 bits in use, and the remaining bits are reserved for future extensions. One instruction

consists of instruction fields that control various stages of execution. These fields are roughly grouped into the 4 DWords as follows:

- Instruction Operation Doubleword (DW0) contains the Opcode and other general instruction control fields.
- Instruction Destination Doubleword (DW1) specifies the destination operand (dst) and the register file and type of source operands.
- Instruction Source 0 Doubleword (DW2) contains the first source operand (src0).
- Instruction Source 1 Doubleword (DW3) contains the second source operand (src1) and is used to hold any 32-bit immediate source (Imm32 as src0 or src1).

Instructions with one source operand of type DF, Q, or UQ can use an Imm64 64-bit immediate source operand in DWord 2 and DWord3.

Most instructions have 1 or 2 source operands and use a common instruction format. Within that format, there are variations based on AddrMode and AccessMode. There is a separate instruction format for a small number of instructions with 3 source operands. Send, math, and branching instructions have format variations described separately.

The 3-source instructions have the following restrictions:

- Only GRF registers can be sources and only GRF registers can be the destination.
- Subregister numbers have DWord granularity if AccessMode is Align16.
- AccessMode is Align16, uses Align16-style swizzling, with extra replication control. There is no other regioning support.

The next two subsections describe the instruction formats for various processor generations using tables. The following diagrams provide another view of the same information. The first diagram is for native instructions with one or two source operands.

GEN Instruction Format – 1-src and 2-src

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	AddrMode = Direct: AccessMode = Align16	AddrMode = Direct: AccessMode = Align1	AddrMode = Indirect: AccessMode = Align16	AddrMode = Indirect: AccessMode = Align1	SEND: MsgDesc Imm	SEND: MsgDesc Reg	MATH	Branch (2offsets)	Branch (1offset)	Imm Src[31:0]	Imm Src[63:0]
	1	127	127	1					EOT						
	6	126	121	6											
	4	120	117	4	Src1.VertStride										
	1	116	116	1											
	2	115	114	2		Src1.Width		Src1.Width							
	2	113	112	2	Src1.ChanSel[7:4]	Src1.HorzStride	Src1.ChanSel[7:4]	Src1.HorzStride							
	1	111	111	1	Src1.AddrMode										
	2	110	109	2	Src1.SrcMod										
	4	108	105	4			Src1.AddrSubRegNum								
	4	104	101	4	Src1.RegNum [7:0]										
	1	100	100	1	Src1.SubRegNum [4]		Src1.AddrImm[8:4]								
3	4	99	96	4	Src1.ChanSel[3:0]	Src1.SubRegNum [4:0]	Src1.ChanSel[3:0]	Src1.AddrImm[8:0]			Same	JIP[31:0]	JIP[31:0]	Imm[31:0]	
	1	95	95	1											
	4	94	91	4	Src1.SrcType										
	2	90	89	2	Src1.RegFile										
	4	88	85	4	Src0.VertStride										
	1	84	84	1											
	2	83	82	2		Src0.Width		Src0.Width							
	2	81	80	2	Src0.ChanSel[7:4]	Src0.HorzStride	Src0.ChanSel[7:4]	Src0.HorzStride							
	1	79	79	1	Src0.AddrMode										
	2	78	77	2	Src0.SrcMod										
	4	76	73	4			Src0.AddrSubRegNum								
	4	72	69	4	Src0.RegNum [7:0]		Src0.AddrImm [8:4]								
	1	68	68	1	Src0.SubRegNum [4]										
2	4	67	64	4	Src0.ChanSel[3:0]	Src0.SubRegNum [4:0]	Src0.ChanSel[3:0]	Src0.AddrImm [8:0]				UIP[31:0]			Imm[63:0]
	1	63	63	1	Dst.AddrMode										
	2	62	61	2		Dst.HorzStride		Dst.HorzStride							
	4	60	57	4			Dst.AddrSubRegNum								
	4	56	53	4	Dst.RegNum [7:0]										
	1	52	52	1	Dst.SubRegNum [4]		Dst.AddrImm [8:4]								
	4	51	48	4	Dst.ChanEn[3:0]	Dst.SubRegNum [4:0]	Dst.ChanEn[3:0]	Dst.AddrImm [8:0]							
	1	47	47	1	Reserved										
	4	46	43	4	Src0.SrcType										
	2	42	41	2	Src0.RegFile										
	4	40	37	4	Dst.Type										
	2	36	35	2	Dst.RegFile										
	1	34	34	1	WECtrl										
	2	33	32	2	FlagRegNum/FlagSubRegNum				Same		Same	Same	Same	Same	Same
	1	31	31	1	Saturate										
	1	30	30	1	DebugCtrl										
	1	29	29	1	CmptCtrl										
	1	28	28	1	AccWrCtrl / BranchCtrl				Same		Same	Same	Same		
	4	27	24	4	CondModifier				SFID[3:0]		FC[3:0]	MBZ	MBZ		
	3	23	21	3	ExecSize										
	1	20	20	1	PredInv										
	4	19	16	4	PredCtrl										
	2	15	14	2	ThreadCtrl										
	2	13	12	2	QtrCtrl										
	1	11	11	1	NibCtrl										
	2	10	9	2	DepCtrl										
	1	8	8	1	AccessMode				Same		Same	Same	Same	Same	Same
	1	7	7	1	(reserved for Opcode)										
0	7	6	0	7	Opcode				Same		Same	Same	Same	Same	Same

The next diagram is for native instructions with three source operands.

GEN Instruction Format – 3-src

DW#	Instr Bits	High Bit	Low Bit	Instr Bits	Description
	1	127	127	1	*reserved*
	1	126	126	1	*reserved*
	8	125	118	8	Src2 Regnum
	3	117	115	3	Src2 Subregnum
	8	114	107	8	Src2 Swizzle
	1	106	106	1	Src2 RepCtrl
	1	105	105	0	*reserved*
	8	104	97	8	Src1 Regnum
	3	96	94	3	Src1 Subregnum
	8	93	86	8	Src1 Swizzle
	1	85	85	1	Src1 RepCtrl
	1	84	84	0	*reserved*
	8	83	76	8	Src0 Regnum
	3	75	73	3	Src0 Subregnum
	8	72	65	8	Src0 Swizzle
2,3	1	64	64	1	Src0 RepCtrl
	8	63	56	8	Dst Regnum
	3	55	53	3	Dst Subregnum
	4	52	49	4	Dst chan enable
	3	48	46	3	Dst Type
	3	45	43	3	Src Type
	2	42	41	2	Src2 Modifier
	2	40	39	2	Src1 Modifier
	2	38	37	2	Src0 Modifier
	1	36	36	1	*reserved*
	1	35	35	1	*reserved*
	1	34	34	1	*WECtrl*
1	2	33	32	2	*FlagRegNum/FlagSubRegNum*
	1	31	31	1	*Saturate*
	1	30	30	1	*DebugCtrl*
	1	29	29	1	*CmptCtrl*
	1	28	28	1	*AccWrCtrl*
	4	27	24	4	*CondModifier*
	3	23	21	3	*ExecSize*
	1	20	20	1	*PredInv*
	4	19	16	4	*PredCtrl*
	2	15	14	2	*ThreadCtrl*
	2	13	12	2	*QtrCtrl*
	1	11	11	1	*NibCtrl*
	2	10	9	2	*DepCtrl*
	1	8	8	1	*AccessMode*
	1	7	7	0	*(reserved for Opcode)*
0	7	6	0	7	*Opcode*

GEN Instruction Format – 3-src

DW#	Instr Bits	High Bit	Low Bit	Instr Bits	Description
	1	127	127	1	*reserved*
	1	126	126	1	Src2 Subregnum
	8	125	118	8	Src2 Regnum
	3	117	115	3	Src2 Subregnum
	8	114	107	8	Src2 Swizzle
	1	106	106	1	Src2 RepCtrl
	1	105	105	1	Src1 Subregnum
	8	104	97	8	Src1 Regnum
	3	96	94	3	Src1 Subregnum
	8	93	86	8	Src1 Swizzle
	1	85	85	1	Src1 RepCtrl
	1	84	84	1	Src0 Subregnum
	8	83	76	8	Src0 Regnum
	3	75	73	3	Src0 Subregnum
	8	72	65	8	Src0 Swizzle
2,3	1	64	64	1	Src0 RepCtrl
	8	63	56	8	Dst Regnum
	3	55	53	3	Dst Subregnum
	4	52	49	4	Dst chan enable
	3	48	46	3	Dst Type
	3	45	43	3	Src Type
	2	42	41	2	Src2 Modifier
	2	40	39	2	Src1 Modifier
	2	38	37	2	Src0 Modifier
	1	36	36	1	Src1 Type
	1	35	35	1	Src2 Type
	1	34	34	1	*WECtrl*
1	2	33	32	2	*FlagRegNum/FlagSubRegNum*
	1	31	31	1	*Saturate*
	1	30	30	1	*DebugCtrl*
	1	29	29	1	*CmptCtrl*
	1	28	28	1	*AccWrCtrl*
	4	27	24	4	*CondModifier*
	3	23	21	3	*ExecSize*
	1	20	20	1	*PredInv*
	4	19	16	4	*PredCtrl*
	2	15	14	2	*ThreadCtrl*
	2	13	12	2	*QtrCtrl*
	1	11	11	1	*NibCtrl*
	2	10	9	2	*DepCtrl*
	1	8	8	1	*AccessMode*
	1	7	7	0	*(reserved for Opcode)*
0	7	6	0	7	*Opcode*

Instruction Fields

This section contains two large tables that together describe all instruction fields. The first table describes common instruction fields that are not specific to a source or destination operand. The second table describes fields used to describe source or destination operands, including fields that describe register regioning and immediate operand fields.

Notable changes for BDW+ instructions include adding three new data types (Q, UQ, HF), widening all type fields to four bits each to accomodate the new type encodings, providing a new source modifier interpretation for logical operations, widening the AddrSubRegNum fields to four bits to accomodate 16 (rather than 8) address subregisters, and supporting 64-bit immediate (Imm64) source values for instructions with single source operands.

In the assembler syntax, some fields appear in the positions used for destination or source operands but are not normal operands. Such fields appear in the Common Instruction Fields table, notably the JIP and UIP instruction offsets used in some flow control instructions.

Common Instruction Fields (Alphabetically by Short Name)

Field	Description
AccessMode	**Access Mode.** This field controls operand access for the instruction. It applies to all source and destination operands. When it is clear (**Align1**), the instruction uses byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are not supported. Elements must still be aligned on element-size boundaries. For example, a DWord operand must be aligned on a 4-byte boundary. When it is set (**Align16**), the instruction uses 16-byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are supported in this mode. A register region must start on 16-byte aligned address, but individual elements do not. For example, a packed array of 16 bytes can be an operand, where only the first byte, at offset 0, is at a 16-byte aligned address. 0 = **Align1** 1 = **Align16**
AccWrCtrl/ BranchCtrl	**Accumulator Write Control.** Enable or disable implicitly writing results to the accumulator as well as to the destination. 0 = Do not write results to the accumulator; write results only to the destination. 1 = **AccWrEn.** Write results to the accumulator as well as to the destination. This bit should not be set if the accumulator is the explicit destination operand. Some instructions do not allow this option, including *mov* and *send*. **Branch Control.** Used by the *goto* instruction to control branching. See the **goto** instruction description for more information about BranchCtrl.
CmptCtrl	**Compaction Control.** Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information. 0 = No Compaction. No compaction. 128-bit native instruction supporting all instruction options. 1 = **CmptCtrl**. Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

Field	Description
CondModifier	**Condition Modifier.** This field sets the flag register based on internal conditional signals output from the execution pipe such as sign, zero, overflow, NaNs, etc. If this field is 0000b, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field applies to all instructions except send, sendc, and math. 0000b = Do not modify the flag register (normal) 0001b = Zero or Equal (**.z** or **.e**) 0010b = Not Zero or Not Equal (**.nz** or **.ne**) 0011b = Greater than (**.g**) 0100b = Greater than or equal (**.ge**) 0101b = Less than (**.l**) 0110b = Less than or equal (**.le**) 0111b = Reserved 1000b = Overflow (**.o**) 1001b = Unordered with Computed NaN (**.u**) 1010b-1111b = Reserved
DepCtrl	**Destination Dependency Control.** This field selectively disables destination dependency check and clear for this instruction. When it is set to 00b, normal destination dependency control is performed for the instruction; hardware checks for destination hazards to ensure data integrity. Specifically, a destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard is cleared when the destination operands retire. When **NoDDClr** is set, the destination register scoreboard is *not* cleared when destination operands retire. When **NoDDChk** is set, hardware does not check for destination register dependencies before the instruction is made ready for execution. **NoDDClr** and **NoDDChk** are not mutually exclusive; both can be set. When this field is not 00b, hardware does not protect against destination hazards for the instruction. These settings are typically used to assemble data in a fine grained fashion (for example, a matrix-vector compute with dot-product instructions), where data integrity is guaranteed by software based on the intended usage of instruction sequences. 00b = Destination dependency checked and cleared (normal). 01b = **NoDDClr**. Destination dependency checked but not cleared. 10b = **NoDDChk**. Destination dependency not checked but cleared. 11b = **NoDDClr, NoDDChk**. Destination dependency not checked and not cleared. See the Destination Hazard and the Use of NoDDClr sections for more information.

Field	Description
EOT	**End of Thread.** For a *send* or *sendc* instruction, this bit controls thread termination. It is not used for other instructions. For a *send* or *sendc*instruction, if this field is set, the EU terminates the thread and also sets the EOT bit in the message sideband. 0 = The thread is not terminated. 1 = End of thread (EOT).
ExDesc	**Extended message descriptor.** A 32 bit immediate extended message descriptor for *send* and *sendc* instructions. This field is not used for other instructions.
ExecSize	**Execution Size.** Specifies the number of parallel execution channels and data elements for the instruction, a power of 2 from 1 to 32. The size cannot exceed the maximum number of channels allowed for the largest data type specified for a source or destination operand. *ExecSize* x largest data type size in bytes <= 64. 000b = 1 Channels (scalar operation). 001b = 2 Channels. Any data type. 010b = 4 Channels. Any data type. 011b = 8 Channels. Any data type. 100b = 16 Channels. 4-byte or smaller data types. Excludes DF, Q, and UQ types. 101b = 32 Channels. 2-byte or 1-byte data types. Excludes D, DF, F, Q, UD, and UQ types. 110b-111b = Reserved. An operand's *Width* must be <= *ExecSize*.
FC	**Function Control.** Specifies the extended math function carried out by the *math*instruction. Not used for any other instruction. This field is in the same position as the CondModifier instruction field, so the *math* instruction does not support conditional modifiers. See the *math* Extended Math Function instruction description for more information about this field.
FlagRegNum/ **FlagSubRegNum**	**Flag Register Number/Flag Sub-Register Number.** Selects the 32-bit flag register (f0 or f1) and the 16-bit flag subregister (.0 or .1). The specified flag subregister is the source for any predication and the destination for new flag values produced by any enabled conditional modifier. A flag subregister can be both a predication source and a conditional modifier destination in the same instruction. The number of flag bits used or updated depends on the execution size. 00b f0.0 01b f0.1 10b f1.0 11b f1.1
JIP	**Jump Instruction Pointer.** A Doubleword Signed Integer offset relative to the current IP (which references the current instruction) in units of bytes. Typically an immediate value in the instruction. For the *brc* (Branch Converging) instruction, both JIP and UIP can be contained in a register.

Field	Description
MaskCtrl	**Mask Control** (formerly WECtrl/Write Enable Control). This flag disables the normal write enables; it should normally be 0. 0 = Use the normal write enables in Dst.ChanEn (normal setting). 1 = NoMask. Write all channels except those disabled by predication or by other masks besides the write enables. MaskCtrl = NoMask also skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.
NibCtrl	**Nibble Control.** This field is used in some instructions along with QtrCtrl. See the description of QtrCtrl below. NibCtrl is only used for SIMD4 instructions with a 64bit datatype as source or destination. 0 = Use an odd 1/8th for DMask/VMask and ARF (first, third, fifth, or seventh depending on QtrCtrl). 1 = Use an even 1/8th for DMask/VMask and ARF (second, fourth, sixth, or eighth depending on QtrCtrl). Note that if eighths are given zero-based indices from 0 to 7, then NibCtrl = 0 indicates even indices and NibCtrl = 1 indicates odd indices.
NoSrcDepSet	**No Source Dependency Set.** In *send*, *sendc*, *sends* and *sendsc* instruction, this bit controls the setting of GRF source dependency. The source dependencies of both sources are considered together. When it is set to 0, normal write after read (WAR) dependency control is performed for the instruction; hardware checks for write after read hazard and ensure data integrity. Specifically source register dependency is set by the current instruction and a destination register dependency check is expected to be conducted before the next instruction is made ready for execution. When it is set to 1, hardware does not protect against write after read data hazard for the instruction. This setting is used in send instruction where the same source data is reused without modification by multiple send instruction. Software must send to the same shared function which must not be in fault and stream page fault mode and guarantee one of the following for all the source registers used in a send instruction to ensure data integrity. • The last send instruction before the instruction which modifies its source(s) does not use this setting. • The instruction which modifies the source(s) is either dependent on the destination register of the last send instruction with the source(s) or has instruction before it which does so. • The instruction which modifies the source(s) is either dependent on the source register of the last send instruction with the source(s) which does not use this setting or has instruction before it which does so. 0 = source dependency is set (normal). 1 = NoSrcDepSet. Source dependency is not set.

Field	Description
OpCode	**Operation Code.** Contains the instruction operation code. Each opcode used is given a unique mnemonic. For example, the opcode 0x01 has the mnemonic *mov* indicating a Move instruction. See the Opcode Encoding section for details of opcode encoding.
PredCtrl	**Predicate Control.** This field, together with PredInv, controls generating the predication mask (PMask) for the instruction. It allows per-channel conditional execution the instruction based on the content of the selected flag register. The encoding depends on the access mode. See the Predication section for more information about predication. In **Align16** access mode, there are eight encodings (including no predication). All encodings are based on group-of-4 predicate bits, including channel sequential, replication swizzles and horizontal any or all operations. The same configuration is repeated for each group-of-4 execution channels. In **Align1** access mode, there are twelve encodings (including no predication). The encodings apply to all execution channels with explicit channel grouping from a single channel up to a group of 16 channels. *Predicate Control in* **Align16** *access mode:* 0000b = No predication (normal). 0001b = Predication with sequential flag channel mapping. 0010b = Predication with replication swizzle **.x**. 0011b = Predication with replication swizzle **.y**. 0100b = Predication with replication swizzle **.z**. 0101b = Predication with replication swizzle **.w**. 0110b = Predication with **.any4h.** 0111b = Predication with **.all4h.** 1000b-1111b = Reserved. *Predicate Control in* **Align1** *access mode:* 0000b = No predication (normal). 0001b = Predication with sequential flag channel mapping. 0010b = Predication with **.anyv** (f0.0 OR f1.0 on the same channel). 0011b = Predication with **.allv** (f0.0 AND f1.0 on the same channel). 0100b = Predication with **.any2h** (any in group of 2 channels). 0101b = Predication with **.all2h** (all in group of 2 channels). 0110b = Predication with **.any4h** (any in group of 4 channels). 0111b = Predication with **.all4h** (all in group of 4 channels). 1000b = Predication with **.any8h** (any in group of 8 channels).

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>1001b = Predication with .all8h (all in group of 8 channels).
1010b = Predication with .any16h (any in group of 16 channels).
1011b = Predication with .all16h (all in group of 16 channels).
1100b = Predication with .any32h (any in group of 32 channels).
1101b = Predication with .all32h (all in group of 32 channels).
1110b-1111b = Reserved.</td></tr>
<tr><td>PredInv</td><td>Predicate Inverse. Together with PredCtrl, controls generation of the predication mask (PMask) for the instruction. When it is set and PredCtrl is not 0000b, predication uses the inverse of the predication bits produced by PredCtrl. In other words, the effect of PredInv happens after the effects of PredCtrl.
This field is ignored if PredCtrl is 0000b; there is no predication.
0 = +. Positive polarity of predication. Use the predication mask produced by PredCtrl.
1 = -. Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
PMask is the final predication mask produced by the effects of both fields.</td></tr>
<tr><td>QtrCtrl</td><td>Quarter Control. This field provides explicit control for ARF selection.
This field combined with ExecSize determines which channels are used for the ARF registers.
Along with NibCtrl, 1/8 DMask/VMask and ARF can be selected.
<table>
<tr><th>QtrCtrl</th><th>NibCtrl</th><th>ExecSize</th><th>Syntax</th><th>Description</th></tr>
<tr><td>00b</td><td>0</td><td>8</td><td>1Q</td><td>Use first quarter for DMask/VMask.
Use first half for everything else.</td></tr>
<tr><td>01b</td><td>0</td><td>8</td><td>2Q</td><td>Use second quarter for DMask/VMask.
Use second half for everything else.</td></tr>
<tr><td>10b</td><td>0</td><td>8</td><td>3Q</td><td>Use third quarter for DMask/VMask.
Use first half for everything else.</td></tr>
<tr><td>11b</td><td>0</td><td>8</td><td>4Q</td><td>Use fourth quarter for DMask/VMask.
Use second half for everything else.</td></tr>
<tr><td>00b</td><td>0</td><td>16</td><td>1H</td><td>Use first half for DMask/VMask.
Use all channels for everything else.</td></tr>
<tr><td>10b</td><td>0</td><td>16</td><td>2H</td><td>Use second half for DMask/VMask.
Use all channels for everything else.</td></tr>
<tr><td>00b</td><td>0</td><td>4</td><td>1N</td><td>Use first 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>00b</td><td>1</td><td>4</td><td>2N</td><td>Use second 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>01b</td><td>0</td><td>4</td><td>3N</td><td>Use third 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>01b</td><td>1</td><td>4</td><td>4N</td><td>Use fourth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>10b</td><td>0</td><td>4</td><td>5N</td><td>Use fifth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>10b</td><td>1</td><td>4</td><td>6N</td><td>Use sixth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>11b</td><td>0</td><td>4</td><td>7N</td><td>Use seventh 1/8th for DMask/VMask and ARF.</td></tr>
</table></td></tr>
</table>

Field	Description
	<table><tr><td>11b</td><td>1</td><td>4</td><td>8N</td><td>Use eighth 1/8th for DMask/VMask and ARF.</td></tr></table> **2H** is only allowed for SIMD16 instruction in Single Program Flow mode (SPF = 1). NibCtrl is only allowed for SIMD4 instructions with a 64bit datatype as source or destination type.
Reg32	In a *send*, *sendc*, *sends* or *sendsc* instruction refers to the option of providing the message descriptor field DWords correspondingly as the first dwords of the Address Register rather than as an immediate operand.
Saturate	**Saturate.** Enables or disables destination saturation. When it is set, output values to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any value outside the saturation target range for the data type to the closest value in the target range. For a floating-point destination type, the saturation target range is [0.0, 1.0]. For a floating-point NaN, there is no "closest value"; any NaN saturates to 0.0. Note that enabling Saturate overrides all of the NaN propagation behaviors described for various numeric instructions. Any floating-point number greater than 1.0, including +INF, saturates to 1.0. Any negative floating-point number, including -INF, saturates to 0.0. Any floating-point number in the range 0.0 to 1.0 is not changed by saturation. For an integer destination type, the maximum range for that type is the saturation target range. For example, the saturation range for B (Signed Byte Integer) is [-128, 127]. When Saturate is clear, destination values are not saturated. For example, a wrapped result (modulo) is output to the destination for an overflowed integer value. See the Numeric Data Types section for information about data types and their ranges. 0 = No destination modification (normal). 1 = **sat**. Saturate the output.
SFID	**Shared Function ID.** Specifies a shared function that is the target of a *send*or *sendc* instruction. This field is not used for any other instructions. This field is in the same position as the CondModifier instruction field, so the *send* and *sendc*instructions do not support conditional modifiers.
ThreadCtrl	**Thread Control.** This field provides explicit control for thread switching. If this field is set to 00b, it is up to the Execution Unit to manage thread switching. This is the normal operating mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. For another example, if the current instruction is ready to run but another thread with higher priority also has an instruction ready, the EU switches to that thread. If this field is **Switch**, a forced thread switch occurs after the current instruction executes and before the next instruction. In addition, a long delay (longer than the execution pipe latency) for the current thread is introduced for the thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. If this field is **Atomic**, the next instruction gets highest priority in the thread arbitration for the execution pipelines.

Field	Description
	Switch is designed primarily as a safety feature in case there are race conditions for certain instructions. 00b = Normal Thread Control. Execution may or may not be preempted by another thread following this instruction. 01b = **Atomic**. Prevent any thread switch immediately following this instruction. Always execute the next instruction (which may not be next sequentially if the current instruction branches). Atomic can be used with send, sends, sendc, sendsc ONLY. 10b = **Switch**. Force a switch to another thread after this instruction and before the next instruction. 11b = Reserved
UIP	**Update Instruction Pointer.** A Doubleword Signed Integer offset relative to the current IP (which references the current instruction) in units of bytes. Typically an immediate value in the instruction. For the *brc* (Branch Converging) instruction, both JIP and UIP can be contained in a register.

In the following table of operand fields, use just the last part of the field name. For example, to find the Src1.ChanSel field, look for ChanSel.

Source or Destination Operand Fields (Alphabetically by Short Name)

Field	Description
AddrImm	**Address Immediate.** A 10-bit signed integer offset in units of bytes, only used with the Indirect Addressing Mode. In that addressing mode, the Address Immediate value is added to an address subregister value to determine the operand's address in the GRF. Allowed for any GRF register operand, destination or source. ARF registers cannot be accessed with indirect addressing. The Address Immediate field cannot cover the 4K-byte range of the thread's GRF. Whatever address subregister value is used along with this offset provides a window into the GRF, limited by the offset range. In the Align16 Access Mode, the low four bits of AddrImm are zero and do not appear in the instruction format. Format = S9 Range = [-512, 511]
AddrMode	**Addressing Mode.** Whether the destination register and subregister are encoded in the instruction (Direct Addressing) or calculated using the contents of an address subregister and an offset (Indirect Addressing). This field applies to source and destination register operands, for instructions with 1 or 2 source operands. 0 = Direct Addressing ("Direct"). Direct register addressing. 1 = Indirect Addressing ("Register-Indirect" or "Indirect"). Register-indirect register addressing. Instructions with 3 source operands use Direct Addressing.

Field	Description
AddrSubRegNum	**Address Sub-Register Number.** The address register contains 16 Word-sized subregisters. This field is only used with the Indirect Addressing Mode and specifies the address subregister containing a value added to the Address Immediate value to determine the operand address. Format = U4 Range = [0, 15] Some variations of Indirect Addressing use multiple address subregisters, where AddrSubRegNum determines the first subregister used.
ChanEn	**Channel Enable.** Dst.ChanEn, used only for the destination operand and only in the Align16 Access Mode. Provides four channel enable bits applied modulo four to all *ExecSize* channels. For example, 0xF enables all channels, 0 disables all channels, 0xA enables odd-numbered channels, and so on. The assembler mnemonics are **x**, **y**, **z**, and **w** for channels 0, 1, 2, and 3 respectively. If MaskCtrl is 1 (mnemonic **NoMask**) then all channels are enabled regardless of the ChanEn value, equivalent to ChanEn of 0xF (**xyzw**). Predication and execution masking, in addition to ChanEn and MaskCtrl, determine what channels are actually written. For i in 0, 1, 2, 3: Bit i = 0 For channel j where j % 4 == i, disable writing that channel. Bit i = 1 For channel j where j % 4 == i, enable writing that channel.
ChanSel	**Channel Select.** This field controls the channel swizzle for a non-immediate source operand in the Align16 access mode. It is not used for immediate operands, destination operands, or in the Align1 access mode. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3:2], [5:4] and [7:6] are for channel 0 (**x**), 1 (**y**), 2 (**z**), and 3 (**w**) in the group, respectively. When operating on 64-bit operands, these channel selects must be used in pairs to select a contiguous 64-bit source. For example with an execution size of 8, r0.0<4>.zywz:f assigns the channels as follows: Chan0 = Data2, Chan1 = Data1, Chan2 = Data3, Chan3 = Data2; Chan4 = Data6, Chan5 = Data5, Chan6 = Data7, Chan7 = Data6. The 2-bit Channel Selection field for each channel within the group of 4 is defined as: 00b = **x**. Channel 0 is selected for the corresponding execution channel. 01b = **y**. Channel 1 is selected for the corresponding execution channel. 10b = **z**. Channel 2 is selected for the corresponding execution channel. 11b = **w**. Channel 3 is selected for the corresponding execution channel. Note: When using channel select for 64-bit operands, the valid selects are **.xy** and **.zw**. This is required to pick a pair of DWords.
Desc	**Message descriptor.** A 31 bit immediate message descriptor for *send*, *sendc*, *sends* and *sendsc* instructions.This field is not used for other instructions.
DstType	**Destination Type.** Dst.DstType specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a *send* or *sendc* instruction, this field applies to CurrDst, the current destination operand. Three source instructions use a 3-bit encoding that allows fewer data types.

Field	Description
ExDesc	**Extended message descriptor.** A 32 bit immediate extended message descriptor for *send*, *sendc*, *sends* and *sendsc* instructions.This field is not used for other instructions.
HorzStride	**Horizontal Stride.** Is the distance in units of data element size between two adjacent data elements within a row (horizontal) in the register region for an operand. A horizontal stride of 0 is used for a row that is one-element wide, useful when an instruction repeats a column value or repeats a scalar value. For example, adding a single column to every column in a 2D array or adding a scalar to every element in a 2D array uses HorzStride of 0. A horizontal stride of 1 indicates that elements are adjacent within a row. References to HorzStride in this volume normally reference the value not the encoding, so there are references to HorzStride of 4, which is encoded as 11b. This field applies to both source and destination register operands. This field is used with both direct and indirect addressing. 00b = 0 Elements 01b = 1 Element 10b = 2 Elements 11b = 4 Elements See the Register Region Restrictions section for rules that constrain *HorzStride* in relation to other region parameters.
Imm[28:0]	A 29-bit immediate message descriptor for a *send* or *sendc* instruction. This field is not used for other instructions.
Imm32	A 32-bit immediate data field for an immediate source operand. Only one source operand can be immediate, the last source operand. Of course a destination operand is never immediate. For a two-source instruction, src1 can be immediate; for a one-source instruction src0 can be. The source type for an immediate operand cannot be B or UB (signed or unsigned byte). The source type can be one of the packed vector types that are only allowed as immediate operands: V, UV, or VF. For the W or UW (signed or unsigned word) source types, the 16-bit value must be replicated in both the low and high words of the 32-bit immediate value. The low order bits are directly used when fewer than 32-bits are needed for the source type. The 32-bit value is not coerced into the designated type. See the Numeric Data Types section for information about data types and their ranges. In *send*, *sendc*, *sends* or *sendsc* instruction, refers to the 31bit immediate message descriptor field and the 32bit immediate extended message descriptor field.
Imm64	A 64-bit immediate data field for an immediate source operand, only used for a one-source instruction with a 64-bit Source Type (DF, Q, or UQ).
RegFile	**Register File.** Select a source or destination register file or indicate an immediate source operand: 00b = ARF Architecture Register File. Only allowed for src0 or destination. 01b = GRF General Register File. Allowed for any source or destination. 10b = Reserved. 11b = Immediate operand. Only allowed for the last source operand. Not allowed for the

Field	Description
	destination operand or for any other source operand. Note that for flow control instructions requiring two offsets, regfile of source0 is required to be immediate since the 64b for immediates occupy the DW2 and DW3.
RegNum	**Register Number.** The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number. An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers. RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands. Format = U8, if RegFile = GRF. 0x00 to 0x7F = Register number in the range [0, 127]. 0x80 to 0xFF = Reserved. Format = 8-bit encoding, if RegFile = ARF. This field encodes the architecture register type as well as providing the register number. See the ARF Registers Overview section and the sections for individual ARF registers for details.
RepCtrl	**Replicate Control.** This field is only present in three-source instructions, for each of the three source operands. It controls replication of the starting channel to all channels in the execution size. ChanSel does not apply when Replicate Control is set. This is applicable to 32b datatypes and 16b datatype. 64b datatypes cannot use the replicate control. 0 = No replication. 1 = Replicate across all channels.
SelReg32Desc	**Select Reg32 for message descriptor.** In *sends* or *sendsc* instruction, refers to the selection of Reg32 for the message descriptor field. 0 = Desc. 1 = Reg32. First Dword of Address Register is used for message descriptor.
SpecialAcc	**Special Accumulator.** Specify the accumulator numbers used by the IEEE macro instructions. The 8 special accumulators, acc2 to acc9 are encoded consecutively from 0000b to 0111b and noacc, indicating no special accumulator used is encoded as 1000b.
SrcMod	**Source Modifier.** Specify any numeric (normally) or logical (for logic instructions) modification to a source value before delivery to the execution pipe. The numeric value of each data element of a source operand can optionally have its absolute value taken, its sign inverted (arithmetic negation), or both (absolute value followed by arithmetic negation producing a guaranteed negative value). When used with logic instructions (*and*, *not*, *or*, *xor*), this field indicates whether the source bits are inverted (bitwise NOT) before delivery to the execution pipe, regardless of the source type. This field only applies to source operands. It does not apply to the destination. This field is not present for an immediate source operand.

Field	Description
	For no modification, there is no assembler notation or syntax. Encoding for all instructions other than logic instructions: 00b = No modification (normal). 01b = **(abs)**. Absolute value. 10b = **-**. Negation. 11b = **-(abs)**. Negation of the absolute value (forced negative value). Encoding for logic instructions: 00b = No modification (normal). 01b = No modification (normal). This encoding cannot be selected in the assembler syntax. 10b = **~** Indicates a bitwise NOT, inverting the source bits. 11b = No modification (normal). This encoding cannot be selected in the assembler syntax.
SrcType	**Source Type.** Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF. Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions. In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW. Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding. The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands. Three source instructions can use operands with mixed-mode precision. When SrcType field is set to :f or :hf it defines precision for source 0 only, and fields Src1Type and Src2Type define precision for other source operands: 0b = **:f**. Single precision Float (32-bit). 1b = **:hf**. Half precision Float (16-bit).

Field	Description
SubRegNum	**Sub-Register Number.** The subregister number for the operand. For a GRF register, is the byte address within a 256-bit (32-byte) register. For an ARF register, determines the sub-register number according to the specified encoding for the given architecture register. RegNum and SubRegNum together provide the byte-aligned address for the origin of a GRF register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. **Note:** The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands. Format = U5, if RegFile = GRF and the instruction has fewer than three source operands. 0x00 to 0x1F = Sub-Register number in the range of [0, 31]. Format = U3, if RegFile = GRF and the instruction has three source operands. 0x0 to 0x7 = Sub-Register number MSBs in the range of [0,7]. The two LSBs are zero. Format = U4, if RegFile = GRF and the instruction has three source operands. 0x0 to 0x15 = Sub-Register number MSBs in the range of [0,15]. Format = 5-bit encoding, if RegFile = ARF. See the ARF Registers Overview section and the sections for individual ARF registers for details.
VertStride	**Vertical Stride.** The vertical stride of a source operand's register region in units of data element size. Supported values are 0, powers of 2 from 1 to 32, and a special encoding used for indirect addressing in Align1 mode. Values greater than 32 are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). The special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (AddrMode = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute source addresses. Each address sub-register provides the origin for a row of data elements. The number of address sub-registers used is equal to instruction's ExecSize / source operand's Width. This field only applies to source operands. It does not apply to the destination. For Align16 access mode, only encodings of 0000b, 0010b, and 0011b are allowed. Other codes are reserved. Note 1: A Vertical Stride larger than 32 is not allowed due to the restriction that a source

Field	Description
	operand must reside within two adjacent 256-bit registers (64 bytes total). Note 2: In Align16 access mode, as encoding 1111b is reserved, only single-index indirect addressing is supported. Note 3: If indirect addressing is supported for src1, the encoding 1111b is reserved for src1 and only single-index indirect addressing is supported. Note 4: The encoding 0010b is used for 64-bit operands (types DF, Q, or UQ). 0000b = 0 Elements 0001b = 1 Element. Align1 mode only. 0010b = 2 Elements 0011b = 4 Elements 0100b = 8 Elements. Align1 mode only. 0101b = 16 Elements (applies to byte or word operands only). Align1 mode only. 0110b = 32 Elements (applies to byte operands only). Align1 mode only. 0111b-1110b = Reserved. 1111b = VxH or Vx1 mode (only valid for register-indirect addressing in Align1 mode).
Width	**Width.** The number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the ExecSize field of the instruction. This field only applies to source operands. It does not apply to the destination. 000b = 1 Element 001b = 2 Elements 010b = 4 Elements 011b = 8 Elements 100b = 16 Elements 101b-111b = Reserved Note that with ExecSize of 32, because the maximum Width is 16, there are at least two rows in a source region.

Native Instruction Layouts

This section describes the Execution Unit instruction formats.

DWord 0, bits 31:0 of the 128-bit instruction, has the same format regardless of the number of source operands.

The following three tables cover the most common instruction format, for instructions with 1 or 2 source operands; then the format for the few instructions with 3 source operands; and finally format variations used by a few specific instructions, including branching instructions.

Execution Unit Instruction Format for 1 or 2 Source Operands

Bits	Description	AddrMode and AccessMode Variations			
		AddrMode = Direct		AddrMode = Indirect	
		Align16	Align1	Align16	Align1
Any Imm32 32-bit or Imm64 64-bit immediate operand uses bits 127:96, replacing the following fields.					
127:122	Reserved				
121	Varies based on AddrMode	Reserved		Src1.AddrImm[9]	
120:117	Src1.VertStride				
116	Varies based on AccessMode	Reserved	Src1.Width	Reserved	Src1.Width
115:114		Src1.ChanSel[7:4]		Src1.ChanSel[7:4]	
113:112			Src1.HorzStride		Src1.HorzStride
111	Src1.AddrMode				
110:109	Src1.SrcMod				
108:105	Src1.SrcMod	Src1.RegNum		Src1.AddrSubRegNum	
104:101				Src1.AddrImm[8:4]	Src1.AddrImm[8:0]
100		Src1.SubRegNum[4]	Src1.SubRegNum[4:0]		
99:96		Src1.ChanSel[3:0]		Src1.ChanSel[3:0]	
Any Imm64 64-bit immediate operand uses bits 127:64, replacing the following fields.					
95	Varies based on AddrMode	Reserved		Src0.AddrImm[9]	
94:91	Src1.SrcType				
90:89	Src1.RegFile				
88:85	Src0.VertStride				
84	Varies based on AccessMode	Reserved	Src0.Width	Reserved	Src0.Width
83:82		Src0.ChanSel[7:4]		Src0.ChanSel[7:4]	
81:80			Src0.HorzStride		Src0.HorzStride
79	Src0.AddrMode				
78:77	Src0.SrcMod				
76:73	Varies based on AddrMode and AccessMode	Src0.RegNum		Src0.AddrSubRegNum	
72:69				Src0.AddrImm[8:4]	Src0.AddrImm[8:0]
68		Src0.SubRegNum[4]	Src0.SubRegNum[4:0]		
67:64		Src0.ChanSel[3:0]		Src0.ChanSel[3:0]	
63	Dst.AddrMode				

Bits	Description	AddrMode and AccessMode Variations			
		AddrMode = Direct		AddrMode = Indirect	
		Align16	Align1	Align16	Align1
62:61	Varies based on AccessMode	Reserved	Dst.HorzStride	Reserved	Dst.HorzStride
60:57	Varies based on AddrMode and AccessMode	Dst.RegNum		Dst.AddrSubRegNum	
56:53				Dst.AddrImm[8:4]	Dst.AddrImm[8:0]
52		Dst.SubRegNum[4]	Dst.SubRegNum[4:0]		
51:48		Dst.ChanEn[3:0]		Dst.ChanEn[3:0]	
47	Varies based on AddrMode	Reserved : MBZ		Dst.AddrImm[9]	
46:43	Src0.SrcType				
42:41	Src0.RegFile				
40:37	Dst.DstType				
36:35	Dst.RegFile				
34	MaskCtrl				
33:32	FlagRegNum / FlagSubRegNum				
31	Saturate				
29	CmptCtrl				
28	AccWrCtrl/BranchCtrl				
27:24	CondModifier				
23:21	ExecSize				
20	PredInv				
19:16	PredCtrl				
15:14	ThreadCtrl				
13:12	QtrCtrl				
11	NibCtrl				
10:9	DepCtrl				
8	AccessMode				
7	Reserved (for future Opcode expansion)				
6:0	Opcode				

The 3-source operand instructions are:

1. *bfe* - Bit Field Extract
2. *bfi2* - Bit Field Insert 2
3. *lrp* - Linear Interpolation
4. *mad* - Multiply Add
5. *madm* - Multiply Add for Macro

In the 3-source instruction format, the upper QWord contains three groups of 21 bits for the three source operands, where each group contains four fields in 20 bits and otherwise adjacent groups are separated by single reserved bits.

Specific instructions have different instruction formats as described below. These instructions include send / sendc, math, and branch instructions.

Execution Unit Instruction Format for Specific Instructions

The instruction fields Src0.VertStride, Src0.Width[1:0], Src0.ChanSel[7:4], Src0.HorzStride, Src0.ChanSel[3:0], Src0.AddrImm[3:0], Src1.SrcType and AccWrCtrl are not used for send/sendc/sends/sendsc instructions. Additionally for sends/sendsc instructions Src0.SrcMod, Dst.HorzStride, Dst.ChanEn, Dst.AddrImm[3:0] and Src0.SrcType are not used.

Src0.RegFile[1], Src1.RegFile[1] and Dst.RegFile[1] are implicitly set to 0, and Src0.RegFile[0] is implicitly set as 1 for sends/sendsc instructions. Also note, ExDesc[4] and ExDesc[16:9] are unused and not encoded below.

Bits	Regular 1 or 2 Source Operands Description	Empty white areas mean Same, use the regular description			
		send / sendc	sends / sendsc	math	Branch Instructions
127	Reserved	ExDesc[5] (EOT)			JIP[31:0]
126:125	Reserved	Desc[30:0] / Reg32			
124:121	Reserved				
120:117	Src1.VertStride				
116:112	Varies based on AccessMode				
111	Src1.AddrMode				
110:109	Src1.SrcMod				
108:96	Varies based on AddrMode and AccessMode				
95	Reserved		ExDesc[31:16]/Reg32		UIP[31:0] (2-offset branches)
94:91	Src1.SrcType	ExDesc[31:28]			
90:89	Src1.RegFile				
88:85	Src0.VertStride	ExDesc[27:24]			
84	Varies based on AccessMode				
83:80	Varies based on AccessMode	ExDesc[23:20]			
79	Src0.AddrMode				
78	Src0.SrcMod[1]		Src0.AddrImm[9]		
77	Src0.SrcMod[0]		SelReg32Desc		
76:68	Varies based on AddrMode and AccessMode				
67:64	Varies based on AddrMode and AccessMode	ExDesc[19:16]	ExDesc[9:6]		

Bits	Regular 1 or 2 Source Operands Description	Empty white areas mean Same, use the regular description			
		send / sendc	sends / sendsc	math	Branch Instructions
63	Dst.AddrMode				
62	Varies based on AccessMode		Dst.AddrImm[9]		
61	Varies based on AccessMode		SelReg32ExDesc		
60:52	Varies based on AddrMode and AccessMode				
51:48	Varies based on AddrMode and AccessMode		Src1.RegNum[7:4]		
47	Reserved		Src1.RegNum[3:0]		
46:44	Src0.SrcType				
43	Src0.SrcType				
42	Src0.RegFile[1]				
41	Src0.RegFile[0]				
40:37	Dst.DstType				
36	Dst.RegFile[1]		Src1.RegFile[0]		
35	Dst.RegFile[0]				
34	MaskCtrl				
33:32	FlagRegNum / FlagSubRegNum				
31	Saturate				
30					
29	CmptCtrl				
28	AccWrCtrl	NoSrcDepSet			
27:24	CondModifier	ExDesc[3:0] (SFID)		FC[3:0]	MBZ
23:21	ExecSize				
20	PredInv				
19:16	PredCtrl				
15:14	ThreadCtrl				
13:12	QtrCtrl				
11	NibCtrl				
10:9	DepCtrl				
8	AccessMode				
7	Reserved (for future Opcode expansion)				
6:0	Opcode				

EU Compact Instructions

On receiving an instruction with bit 29 (CmptCtrl) set, HW recognizes it as a 64-bit compact instruction. Hardware then uses the index fields inside the compact instruction to lookup values in the associated compaction tables, then uses the table outputs along with other fields in the compact instruction to reconstruct the 128-bit native-sized instruction.

Description
All flow control instructions use the new offset format, a signed 32-bit offset in units of bytes.

The native 128-bit instruction format provides access to all instruction options. Only some instruction options and combinations of instruction options can be represented in the compact instruction formats.

Which native instructions can be represented as compact instructions and the details of the compact instruction formats and the compaction tables used may change with each processor generation.

In the following instruction format tables the Mapping Bits and Mapping Description columns describe the mappings into native instruction fields.

Opcode Encoding

Byte 0 of the 128-bit instruction word contains the opcode. The opcode uses 7 bits. Bit location 7 in byte 0 is reserved for future opcode extension.

The opcodes are encoded and organized into five groups based on the type of operations: Special instructions, move/logic instructions (opcode=00xxxxxb), flow control instructions (opcode=010xxxxb), miscellaneous instructions (opcode=011xxxxb), parallel arithmetic instructions (opcode=100xxxxb), and vector arithmetic instructions (opcode=101xxxxb). Opcodes 110xxxb are reserved.

Note: Opcodes appear in the overall Instruction Set Summary Table as well. The following subsections still serve the purpose of describing various instruction groups.

Move and Logic Instructions

This instruction group has an opcode format of 00xxxxxb.

- The opcodes for move instructions (*mov*, *sel* and *movi*) share the common 5 MSBs in the form of 00000xxb.
- The opcodes for logic instructions (*not*, *and*, *or*, and *xor*) share the common 5 MSBs in the form of 00001xxb.
- The opcodes for shift instructions (*shr, shl,* ***asr***, **ror**, and **rol**) share the common 4 MSBs in the form of 0001xxxb.
- The opcodes for compare instructions (*cmp* and *cmpn*) share the common 6 MSBs in the form of 001000xb. Bit 0 indicates whether it is a normal compare, *cmp*, or a special compare-NaN, *cmpn*.

Move and Logic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
1	0x01	mov	Component-wise move	1	1
2	0x02	sel	Component-wise selective move based on predication	2	1
3	0x03	movi	Fast component-wise indexed move	1	1
4	0x04	not	Component-wise one's complement (bitwise not)	1	1
5	0x05	and	Component-wise logical AND (bitwise and)	2	1
6	0x06	or	Component-wise logical OR (bitwise or)	2	1
7	0x07	xor	Component-wise logical XOR (bitwise xor)	2	1
8	0x08	shr	Component-wise logical shift right	2	1
9	0x09	shl	Component-wise logical shift left	2	1
10	0x0A	smov	Scattered Move	1	1
11	0x0B	*Reserved*			
12	0x0C	asr	Component-wise arithmetic shift right	2	1
13	0x0D	*Reserved*			
16	0x10	cmp	Component-wise compare, store condition code in destination	2	1
17	0x11	cmpn	Component-wise compare-NaN, store condition code in destination	2	1
18	0x12	csel	Component-wise selective move based on result of compare	3	1
19	0x13	*Reserved*			
20	0x14	*Reserved*			
21	0x15	*Reserved*			
22	0x16	*Reserved*			
23	0x17	bfrev	Reverse bits	1	1
24	0x18	bfe	Bitfield extract	3	1
25	0x19	bfi1	Bitfield insert macro instruction 1, generate mask	2	1
26	0x1A	bfi2	Bitfield insert macro instruction 2, insert based on mask	3	1
27-31	0x1B-0x1F	*Reserved*			

Flow Control Instructions

This instruction group has an opcode format of 010xxxxb.

Flow Control Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
32	0x20	jmpi	Jump indexed	1	0

Opcode		Instruction	Description	#src	#dst
dec	hex				
33	0x21	brd	Branch - Diverging	1	0
34	0x22	if	If	0/2	0
35	0x23	brc	Branch - Converging	1	-
36	0x24	else	Else	1	0
37	0x25	endif	End if	0	0
38	0x26	*Reserved*			
39	0x27	*Reserved*			
40	0x28	break	Break	1	0
41	0x29	cont	Continue	1	0
42	0x2A	halt	Halt	1	0
43	0x2B	calla	Subroutine call absolute	1	1
44	0x2C	call	Subroutine call	1	1
45	0x2D	return	Subroutine return	1	1
46	0x2E	goto	Goto	2	0
47	0x2F	join	Join	2	0

Miscellaneous Instructions

This instruction group has an opcode format of 011xxxxb.

Miscellaneous Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
48	0x30	wait	Wait for (external) notification	1	0
49	0x31	send	Send	1	1
50	0x32	sendc	Conditional Send (based on TDR)	1	1
51	0x33	sends	Split Send	2	1
52	0x34	sendsc	Conditional Split Send (based on TDR)	2	1
53-55	0x35-0x37	*Reserved*			
56	0x38	math	Math functions for extended math pipeline	1/2	1/2
57-63	0x39-0x3F	*Reserved*			

Parallel Arithmetic Instructions

This instruction group has an opcode format of 100xxxxb.

Parallel Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
64	0x40	add	Component-wise addition	2	1
65	0x41	mul	Component-wise multiply	2	1
66	0x42	avg	Component-wise average of the two source operands	2	1
67	0x43	frc	Component-wise floating point truncate-to-minus-infinity fraction	1	1
68	0x44	rndu	Component-wise floating point rounding up (ceiling)	1	1
69	0x45	rndd	Component-wise floating point rounding down (floor)	1	1
70	0x46	rnde	Component-wise floating point rounding toward nearest even	1	1
71	0x47	rndz	Component-wise floating point rounding toward zero	1	1
72	0x48	mac	Component-wise multiply accumulate	2	1
73	0x49	mach	multiply accumulate high	2	1
74	0x4A	lzd	leading zero detection	1	1
75	0x4B	fbh	Find first 1 for UD from msb side, or first 1/0 for D.	1	1
76	0x4C	fbl	First first 1 for UD from lsb side	1	1
77	0x4D	cbit	Count bits set	1	1
78	0x4E	addc	Integer add with carry	2	1 + acc.
79	0x4F	subb	integer subtract with borrow	2	1 + acc.

Vector Arithmetic Instructions

This instruction group has an opcode format of 101xxxxb.

Vector Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
80	0x50	sad2	2-wide sum of absolute difference	2	1
81	0x51	sada2	2-wide sad accumulate	2	1
82-83	0x52-0x53	*Reserved*			
84	0x54	dp4	4-wide dot product for 4-vector	2	1
85	0x55	dph	4-wide homogenous dot product for 4-vector	2	1
86	0x56	dp3	3-wide dot product for 4-vector	2	1
87	0x57	dp2	2-wide dot product for 4-vector	2	1
88	0x58	*Reserved*			
89	0x59	line	Component-wise line equation computation (a multiply-add)	2	1
90	0x5A	pln	Component-wise floating point plane equation computation (a	2	1

Opcode		Instruction	Description	#src	#dst
dec	hex				
			multiply-multiply-add)		
91	0x5B	fma(mad)	Component-wise floating point mad computation (a multiple-add)	3	1
92	0x5C	lrp	Component-wise floating point lrp computation (blend)	3	1
93	0x5D	***fmam(madm)***	Component-wise floating point fused multiply and add for macro operations.	3	1
94-95	0x5E-0x5F	*Reserved*			

Special Instructions

There are two special instructions, namely, *nop* (opcode = 0x7E) and *illegal* (opcode = 0x00).

- *Nop* instruction may be used for instruction padding in memory between two normal instructions to force alignment or to introduce instruction execution delay. Currently, there is no need for between-instruction padding.
- *Illegal* instruction may be used for instruction padding in memory outside the normal instruction sequence such as before or after the kernel program as well as between subroutines.
- *Nop* and *illegal* instructions do not have source operands or destination operand. Therefore, they do not implicitly update the accumulator register. They cannot be compressed.

Special Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
0	0x00	illegal	Illegal instruction	0	0
96-125	0x60-0x7D	*Reserved*			
126	0x7E	nop	No-op	0	0
127	0x7F	*Reserved*			

Native Instruction BNF

The Backus-Naur Form (BNF) grammar identifies the assembly language syntax, which is native to the hardware. It does not include intelligent defaults, assembler pragmas, etc.

Instruction Groups

<Instruction>::=<UnaryInstruction>

|<BinaryAccInstruction>

|<BinaryInstruction>

|<TriInstruction>

|<JumpInstruction>

|<BranchLoopInstruction>

|<ElseInstruction>

|<BreakInstruction>

|<MaskControlInstruction

|<TriInstruction2>

|<CallInstruction>

|<BranchConvIntruction>

|<BranchDivInstruction>

|<MathInstruction>

|<SyncInstruction>

|<SpecialInstruction>

<UnaryInstruction>::= <Predicate> <UnaryInst> <ExecSize> dst <SrcAccImm> <InstOptions>

<UnaryInst>::= <UnaryOp> <ConditionalModifier> <Saturate>

<UnaryOp>::= "*mov*" | "*frc*" | "*rndu*" | "*rndd*" | "*rnde*" | "*rndz*" | "*not*" | "*lzd*"

<BinaryInstruction>::= <Predicate> <BinaryInst> <ExecSize> dst <Src> <SrcImm> <InstOptions>

<BinaryInst>::= <BinaryOp> <ConditionalModifier> <Saturate>

<BinaryOp>::= "**mul**" | "**mac**" | "**mach**" | "**line**"" | "**pln**"

|"**sad2**" | "**sada2**" | "**dp4**" | "**dph**" | "**dp3**" | "**dp2**"" | "**lrp**"" | "**bfi1**"" | "**addc**"" | "**subb**"

<BinaryAccInstruction>::= <Predicate> <BinaryAccInst> <ExecSize> dst <SrcAcc> <SrcImm> <InstrOptions>

<BinaryAccInst>::= <BinaryAccOp> <ConditionalModifier> <Saturate>

<BinaryAccOp>::= "**avg**" | "**add**" | "**sel**"

|"**and**" | "**or**" | "**xor**"

|"**shr**" | "**shl**" | "**asr**"

|"**cmp**" | "**cmpn**"

<TriInstruction>::= <Predicate> <TriInst> <ExecSize> <PostDst> <CurrDst> <TriSrc> <MsgDesc> <InstOptions>

<TriInst>::= <TriOp> <ConditionalModifier> <Saturate>

<TriOp>::= "**send**"

<TriInstruction2> ::= <Predicate> <TriInst2> <ExecSize> dst <Src> <Src> <Src><InstOptions>

<TriInst2> ::= <TriOp> <ConditionalModifier><Saturate>

,<TriOp> ::= “bfe”| “bfi2”|”mad”

<BranchConvInstruction> ::= <Predicate> <BranchConvOp> <ExecSize>< RelativeLocation2>

<BranchConvOp> ::= “brc”

<BrancConvInstruction> ::= <Predicate> <BranchDivOp> <ExecSize>< RelativeLocation3>

<BranchDivOp> ::= “brd”

<CallInstruction> ::= <Predicate> <CallOp> <ExecSize>dst< RelativeLocation2>

<CallOp> ::= “call” |”CALLA”

<MathInstruction> ::= <Predicate> <MathInst> <ExecSize>< Dst>< Src>< Src><FC>

<MathInst> ::= <MathOp><Saturate>

<MathOp> ::= “math”

<FC> ::= “**INV**” | “**LOG**” | “**EXP**” | “**SQRT**” |”**RSQ**” | “**POW**” | “**SIN**” | “**COS**” | “**INT DIV**”

<JumpInstruction> ::=<JumpOp> <RelativeLocation2>

<JumpOp>::= “**jmpi**”

<BranchLoopInstruction>::= <Predicate> <BranchLoopOp> < RelativeLocation>

<BranchLoopOp>::=“**if**” | “**iff**” | “**while**”

<ElseInstruction>::= <ElseOp> < RelativeLocation>

<ElseOp>::=“**else**”

<BreakInstruction>::= <Predicate> <BreakOp> <LocationStackCtrl>

<BreakOp>::=“**break**” | “**cont**” | “**halt**”

<SyncInstruction>::= <Predicate> <SyncOp> <NotifyReg>

<SyncOp>::=“**wait**”

<SpecialInstruction>::=“**do**” | “**endif**” |”**nop**” | “**illegal**”

Destination Register

dst::=<DstOperand>

|<DstOperandEx>

<DstOperand>::=<DstReg> <DstRegion> <WriteMask> <DstType>

<DstOperandEx>::=<AccReg> <DstRegion> <DstType>

|<FlagReg> <DstRegion> <DstType>

|<AddrReg> <DstRegion> <DstType>

|<MaskReg> <DstRegion> <DstType>

|<MaskStackReg>

|<ControlReg>

|<IPReg>

|<NullReg>

| <ChannelEnableReg>

|<ThreadControlReg>

|<PerformanceReg>

<DstReg>::=<DirectGenReg> | <IndirectGenReg>

|<DirectMsgReg> | <IndirectMsgReg>

<PostDst>::=<PostDstReg> <DstRegion> <WriteMask> <DstType>

|<NullReg>

<PostDstReg>::= <DirectGenReg> | <IndirectGenReg>

<CurrDst>::=<DirectAlignedMsgReg>

Source Register

Source with Accumulator Access and with Immediate

<SrcAccImm>::=<SrcAcc>

|<Imm32> <SrcImmType>

<SrcAcc>::=<DirectSrcAccOperand>

|<IndirectSrcOperand>

<DirectSrcAccOperand>::=<DirectSrcOperand>

|<SrcArcOperandEx>

|<AccReg> <SrcType>

<SrcArcOperandEx>::=<FlagReg> <Region> <SrcType>

|<AddrReg> <Region> <SrcType>

|<ControlReg>

|<StateReg>

|<NotifyReg>

|<IPReg>

|<NullReg>

| <ChannelEnableReg>

|<ThreadControlReg>

|<PerformanceReg>

<IndirectSrcOperand>::=<SrcModifier> <IndirectGenReg> <IndirectRegion> <Swizzle > <SrcType>

Source without Accumulator Access

<Src>::=<DirectSrcOperand>

|<IndirectSrcOperand>

< DirectSrcOperand>::=<SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>

|<SrcArcOperandEx>

<TriSrc>::=<SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>

|<NullReg>

<MsgDesc>::=<ImmDesc>

|<Reg32>

<Reg32>::=<DirectGenReg> <Region> <SrcType>

Source without Accumulator Access or IP Access

<SrcImm>::=<DirectSrcOperand>

|<Imm32> <SrcImmType>

Address Registers

<AddrParam>::=<AddrReg> <ImmAddrOffset>

<ImmAddrOffset>::= ""

| "**,**" <ImmAddrNum>

Register Files and Register Numbers

Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

<DirectGenReg>::=<GenRegFile> <GenRegNum> <GenSubRegNum>

<IndirectGenReg>::=<GenRegFile> "**[**" <AddrParam> "**]**"

<GenRegFile>::="**r**"

<GenRegNum>:: ="**0**"..."**127**"

<GenSubRegNum>:: = “”

| “.0”…”.3” //incase of DF

| “**.0**”…”**.7**”

| “**.0**”…”**.15**”

| “**.0**”…”**.31**”

<DirectMsgReg>::=<DirectAlignedMsgReg> <MsgSubRegNum>

<DirectAlignedMsgReg>::=<MsgRegFile> <MsgRegNum>

<IndirectMsgReg>::=<MsgRegFile> “**[**” <AddrParam> “**]**”

<MsgRegFile>::=“**m**”

<MsgRegNum>:: =“**0**”…”**15**”

<MsgSubRegNum>:: = <GenSubRegNum>

<AddrReg>::=<AddrRegFile> <AddrSubRegNum>

<AddrRegFile>::=“**a0**”

<AddrSubRegNum>:: = “”

| “**.0**” … “**.7**”

<AccReg>::=“**acc**” <AccRegNum><AccSubRegNum>

<AccRegNum>:: =“**0**” | “**1**”

<AccSubRegNum>:: = <GenSubRegNum>

<FlagReg> ::= “f” <FlagRegNum> <FlagSubRegNum>

<FlagRegNum> :: = “0” | “1”

<FlagReg>::=“**f0**” <FlagSubRegNum>

<FlagSubRegNum>:: =“”

| “**.0**”…”.**1**”

<NotifyReg>::=“**n**” <NotifyRegNum>

<NotifyRegNum>:: =“**0**”…”**2**”

<StateReg>::=“**sr0**” <StateSubRegNum>

<StateSubRegNum>:: =“.**0**”… “.**1**”

<ControlReg>::=“**cr0**” <ControlSubRegNum>

<ControlSubRegNum>:: =“.**0**” …”.**2**”

<IPReg>::=“**ip**”

<NullReg>::=“**null**”

<ThreadControlReg> ::= “tdr0”<ThreadCntrlSubRegNum>

<ThreadCntrlSubRegNum> ::= “.0”…”.7”

<PerformanceReg> ::= “tm0”

<ChannelEnableReg> ::= “ce0.0”

Relative Location and Stack Control

<RelativeLocation>::= <imm16>

<RelativeLocation2>::= <imm32> | <reg32>

<RelativeLocation3> ::= <imm16> | <reg32>

<LocationStackCtrl>::=<imm32>

Regions

<DstRegion>::=“**<**”<HorzStride> “**>**”

<IndirectRegion>::=<Region> | <RegionWH> | <RegionV>

<Region>::=“**<**”<VertStride> “**;**” <Width> “**,**” <HorzStride> “**>**”

<RegionWH>::=“**<**” <Width> “**,**” <HorzStride> “**>**”

<RegionV>::=“**<**”<VertStride> “**>**”

<VertStride>::= “**0**” | “**1**” | “**2**” | “**4**” | “**8**” | “**16**” | “**32**”

<Width>::=“**1**” | “**2**” | “**4**” | “**8**” | “**16**”

<HorzStride>::=“**0**” | “**1**” | “**2**” | “**4**”

Types

SrcType
<SrcType> ::= “**:df**” | “**:f**” | “**:ud**” | “**:d**” | “**:uw**” | “**:w**” | “**:ub**” | “**:b**” | “**:uq**” | “**:q**” | “**:hf**”

<SrcImmType> ::= <SrcType> | “**:v**” | “**:vf**” | “**:uv**”

<DstType> ::= <SrcType>

Write Mask

<WriteMask>::= “”

| “.” “**x**” | “.” “**y**” | “.” “**z**” | “.” “**w**”

| “.” “**xy**” | “.” “**xz**” | “.” “**xw**” | “.” “**yz**” | “.” “**yw**” | “.” “**zw**”

| “.” “**xyz**” | “.” “**xyw**” | “.” “**xzw**” | “.” “**yzw**”

| "." "**xyzw**"

Swizzle Control

<Swizzle>::=""

| "." <ChanSel>

| "." <ChanSel> <ChanSel> <ChanSel> <ChanSel>

<ChanSel>::= "**x**" | "**y**" | "**z**" | "**w**"

Immediate Values

<ImmAddrNum>::="-**512**"... "**511**"

<Imm64> ::= "0.0"... "±1.0*2-1024...1023" | "0"..."264-1" | "-263"..."263-1"

<Imm32>::="**0.0**"... "**±1.0*2$^{-128...127}$**" | "**0**"..."**2^{32}-1**" | "-**2^{31}**"..."**2^{31}-1**"

<Imm16>::="**0**"..."**2^{16}-1**" | "-**2^{15}**"..."**2^{15}-1**"

<ImmDesc>::="**0**"..."**2^{32}-1**"

Predication and Modifiers

Instruction Predication

<Predicate>::=""

|"**(**" <PredState> <FlagReg> <PredCntrl> "**)**"

<PredState>::=""

|"**+**"

|"**-**"

<PredCntrl>::=""

|"**.x**" | "**.y**" | "**.z**" | "**.w**"

|"**.any2h**" | "**.all2h**"

|"**.any4h**" | "**.all4h**"

|"**.any8h**" | "**.all8h**"

|"**.any16h**" | "**.all16h**"

|"**.anyv**" | "**.allv**"

| "**.any32h**"| "**.all32h**"

Source Modification

<SrcModifier**>**::=""

|"**-**"

|"**(abs)**"

|"**-**" "**(abs)**"

Instruction Modification

<ConditionalModifier>::=""

|<CondMod> "**.** " <FlagReg>

<CondMod>::="**.z**" | "**.e**"|"**.nz**" | "**.ne**"|"**.g**"|"**.ge**"|"**.l**"|"**.le**"|"**.o**" |"**.r**" |"**.u**"

<Saturate>::=""

|"**.sat**"

Execution Size

<ExecSize>::="**(**" <NumChannels> "**)**"

<NumChannels>::="**1**" | "**2**" | "**4**" | "**8**" | "**16**" | "**32**"

Instruction Options

<InstOptions> ::= ""

| "**{**" <InstOption> "**}**"

| "**{**" <InstOption> <InstOptionEx> "**}**"

<InstOptionEx> ::= ""

| "**,**" <InstOption> <InstOptionEx>

<InstOption> ::= <AccessMode>

| <AccWrCtrl>

| <ComprCtrl>

| <DependencyCtrl>

| <MaskCtrl>

| <SendCtrl>

| <ThreadCtrl>

<AccessMode> ::= "**Align1**" | "**Align16**"

<AccWrCtrl> ::= "**AccWrEn**"

<ComprCtrl> ::= "**SecHalf**" | "**Compr**"

<DependencyCtrl> ::= "**NoDDChk**" | "**NoDDClr**"

<MaskCtrl> ::= "**NoMask**"

<SendCtrl> ::= "**EOT**"

<ThreadCtrl> ::= "**Switch**"

| "**Atomic**"

Note for Assembler: Compression control "**Compr**" has a direct map to the binary instruction word. It may be omitted if the Assembler can determine whether an instruction is compressable.

Instruction Set Summary Tables

The columns in the following tables specify instruction mnemonics, hex opcodes, full names, instruction groups, processor generation (where blank means available for SNB+), the number of source operands, whether the instruction supports predication, any support for source modifiers, an indication of supported data types, whether the instruction supports saturation, and any support for conditional modifiers.

See the separate Accumulator Restrictions table for information about how instructions are allowed to use accumulators.

With a dozen columns in these tables, some terse notation is used, like IVB+ for IVB+ in the Project column. If the Project column is blank, an instruction is supported for SNB+, all generations implemented or designed so far, from Sandy Bridge forward.

N and Y indicate No (no support for a feature) and Yes (full support for a feature) respectively.

A SrcMod (source modifier) value of Y indicates that a numeric source modifier is allowed, optionally specifying abolute value, negation, or a forced negative value. The value N indicates no source modifier support.

SrcMod Information
A SrcMod value of ** indicates a logical source modifier is allowed, optionally inverting all source bits (a NOT operation).

In the Src Types and Dst Type columns, Int means any integer type and * means such an extensive list of types that you must refer to the detailed instruction description.

Byte (B, UB) and QWord (Q, UQ) integer types cannot be mixed in the same instruction.

Instruction Set Summary Table A to B (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
add	40	Addition	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
addc	4E	Integer Addition with Carry	Parallel Arithmetic	2	Y	N	UD	UD	N	Y
and	05	Logic And	Move and Logic	2	Y	**	Int	Int	N	Equality only
asr	0C	Arithmetic Shift Right	Move and Logic	2	Y	Y	Int	Int	Y	Y
avg	42	Average	Parallel Arithmetic	2	Y	Y	B, UB W, UW D, UD	B, UB W, UW D, UD	Y	Y
bfe	18	Bit Field Extract	Move and Logic	3	Y	N	UD, D	UD, D	N	N
bfi1	19	Bit Field Insert 1	Move and Logic	2	Y	N	UD, D	UD, D	N	N

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
bfi2	1A	Bit Field Insert 2	Move and Logic	3	Y	N	UD, D	UD, D	N	N
bfrev	17	Bit Field Reverse	Move and Logic	1	Y	N	UD	UD	N	N
brc	23	Branch Converging	Flow Control	0 or 1	Y	N	D		N	N
brd	21	Branch Diverging	Flow Control	0 or 1	Y	N	D		N	N
break	28	Break	Flow Control	0	Y	N			N	N

Instruction Set Summary Table C to E (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
call	2C	Call	Flow Control	0	Y	N		D, UD	N	N
calla	2B	Call Absolute	Flow Control	0	Y	N		D, UD	N	N
cbit	4D	Count Bits Set	Move and Logic	1	Y	N	UB, UW, UD	UD	N	N
cmp	10	Compare	Move and Logic	2	Y	Y	*	*	N	Y
cmpn	11	Compare NaN	Move and Logic	2	Y	Y	*	*	N	Y
cont	29	Continue	Flow Control	0	Y	N			N	N
csel	12	Conditional Select	Move and Logic	3	N	Y	F	F	Y	Y
dp2	57	Dot Product 2	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dp3	56	Dot Product 3	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dp4	54	Dot Product 4	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dph	55	Dot Product Homogeneous	Vector Arithmetic	2	Y	Y	F	F	Y	Y
else	24	Else	Flow Control	0	N	N			N	N
endif	25	End If	Flow Control	0	N	N			N	N

Instruction Set Summary Table F to L (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
fbh	4B	Find First Bit from MSB Side	Move and Logic	1	Y	N	D, UD	UD	N	N
fbl	4C	Find First Bit from LSB Side	Move and Logic	1	Y	N	UD	UD	N	N
frc	43	Fraction	Parallel Arithmetic	1	Y	Y	F	F	N	Y
goto	2E	Goto	Flow Control	0	Y	N			N	N
halt	2A	Halt	Flow Control	0	Y	N			N	N
if	22	If	Flow Control	0	Y	N			N	N
illegal	00	Illegal	Special	0	N	N			N	N
jmpi	20	Jump Indexed	Flow Control	1	Y	N	D		N	N
join	2F	Join	Flow Control	0	Y	N			N	N
line	59	Line	Vector Arithmetic	2	Y	Y	F	F	Y	Y
lrp	5C	Linear Interpolation	Vector Arithmetic	3	Y	Y	F	F	N	Y
lzd	4A	Leading Zero Detection	Move and Logic	1	Y	Y	D, UD	UD	Y	Y

Instruction Set Summary Table M to P (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
mac	48	Multiply Accumulate	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
mach	49	Multiply Accumulate High	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
mad	5B	Multiply Add	Vector Arithmetic	3	Y	Y	*	*	Y	Y
madm	5D	Multiply Add for Macro	Vector Arithmetic	3	Y	Y	*	*	Y	Y

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
math	38	Extended Math Function	Miscellaneous	2	Y	N	*	*	Y	N
mov	01	Move	Move and Logic	1	Y	Y	*	*	Y	Y
movi	03	Move Indexed	Move and Logic	1	Y	Y	*	*	Y	N
mul	41	Multiply	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
nop	7E	No Operation	Special	0	N	N			N	N
not	04	Logic Not	Move and Logic	1	Y	**	Int	Int	N	Equality only
or	06	Logic Or	Move and Logic	2	Y	**	Int	Int	N	Equality only
pln	5A	Plane	Vector Arithmetic	2	Y	Y	F	F	Y	Y

Instruction Set Summary Table R to X (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
ret	2D	Return	Flow Control	1	Y	N	D, UD		N	N
rndd	45	Round Down	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rnde	46	Round to Nearest or Even	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rndu	44	Round Up	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rndz	47	Round to Zero	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
sad2	50	Sum of Absolute Difference 2	Vector Arithmetic	2	Y	Y	B, UB	W, UW	Y	Y
sada2	51	Sum of Absolute Difference Accumulate 2	Vector Arithmetic	2	Y	Y	B, UB	W, UW	Y	Y
sel	02	Select	Move and Logic	2	Y	Y	*	*	Y	Y
send	31	Send Message	Miscellaneous	1	Y	N	*	*	N	N
sendc	32	Conditional Send Message	Miscellaneous	1	Y	N	*	*	N	N
sends	33	Split Send Message	Miscellaneous	2	Y	N	*	*	N	N
sendsc	34	Conditional Split Send Message	Miscellaneous	2	Y	N	*	*	N	N
shl	09	Shift Left	Move and Logic	2	Y	Y	Int	Int	Y	Y
shr	08	Shift Right	Move and Logic	2	Y	Y	Int	Int	Y	Y
smov	0A	Scattered Move	Move and Logic	1	Y	N	*	*	N	N
subb	4F	Integer Subtraction with Borrow	Parallel Arithmetic	2	Y	N	UD	UD	N	Y
wait	30	Wait	Miscellaneous	1	N	N	UD	UD	N	N
while	27	While	Flow Control	0	Y	N			N	N
xor	07	Logic Xor	Move and Logic	2	Y	**	Int	Int	N	Equality only

Accumulator Restrictions

This section describes restrictions on accumulator access: general restrictions, restrictions for specific instructions, and how those specific restrictions vary for processor generations. See Accumulator Registers

for a description of the accumulator registers.

Accumulator registers can be accessed as explicit source or destination operands, as an implicit source value when specified for a particular instruction (*sada2* for example), and as an implicit destination when the AccWrEn instruction option is used.

These general rules apply to accumulator access:

1. Flow control, *send*, *sendc*, and *wait* instructions cannot use accumulators.

2. Instructions that use the accumulator as an implicit source value cannot specify an explicit accumulator source operand.
3. Instructions that specify an implicit accumulator destination (with AccWrEn) cannot specify an explicit accumulator destination operand.
4. An instruction with both an explicit accumulator source operand and an explicit accumulator destination operand must specify the same accumulator register as the source and the destination.

Description
5. Instructions with three source operands cannot use explicit accumulator operands. AccWrEn may be allowed for implicitly updating the accumulator.

These descriptions are frequently used in this table:

- No restrictions.
- No accumulator access, implicit or explicit.
- Source operands cannot be accumulators.
- Source modifier is not allowed if source is an accumulator.
- Accumulator is an implicit source and thus cannot be an explicit source operand.
- Accumulator cannot be destination, implicit or explicit.
- AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

These minor cases occur occasionally in the table:

- Integer source operands cannot be accumulators.
- No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.
- An accumulator can be a source or destination operand but not both.

A few instructions use more than one of the listed restrictions.

Accumulator Restrictions

Instructions	Accumulator Restrictions
add	No restrictions.
addc	AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
and	Source modifier is not allowed if source is an accumulator.
asr *avg*	No restrictions.
bfe	No accumulator access, implicit or explicit.

Instructions	Accumulator Restrictions
bfi1 *bfi2* *bfrev* *cbit*	
cmp	No restrictions.
cmpn	No restrictions.
csel	No restrictions.
dp2 *dp3* *dp4* *dph*	Source operands cannot be accumulators.
fbh *fbl*	No accumulator access, implicit or explicit.
frc	No restrictions.
line	Source operands cannot be accumulators.
lrp	No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.
lzd	No restrictions.
mac	Accumulator is an implicit source and thus cannot be an explicit source operand.
mach	Accumulator is an implicit source and thus cannot be an explicit source operand. AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
madm	No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.
math	No accumulator access, implicit or explicit.
mov	An accumulator can be a source or destination operand but not both.
movi	Source operands cannot be accumulators.
mul	Integer source operands cannot be accumulators.
not *or*	Source modifier is not allowed if source is an accumulator.
pln	Source operands cannot be accumulators.
rndd	No restrictions.

Instructions	Accumulator Restrictions
rnde *rndu* *rndz*	
sad2 *sada2*	Source operands cannot be accumulators.
sel	No restrictions.
shl	Accumulator cannot be destination, implicit or explicit.
shr	No restrictions.
smov	No restrictions.
subb	AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
xor	Source modifier is not allowed if source is an accumulator.

Instruction Set Reference

This chapter describes the functions of 3D Media GPGPU Execution Units, listed in alphabetical order according to assembly language mnemonic.

EUISA Instructions List

Conventions

This section describes conventions used in instruction reference pages.

For each instruction that has source or destination types, a table lists the allowed type combinations and may also indicate the processor generations that support certain combinations. A notation like *W indicates that UW and W are both allowed. Multiple types listed together mean that any combination (Cartesian product) of the listed types is allowed.

If a source operand is floating-point, all source operands must have the same floating-point data type.

The Q and UQ types cannot be mixed with the B or UB types, neither as different source types nor as source type and destination type.

Accumulator restrictions are described in the Accumulator Restrictions section and also appear in instruction descriptions.

Pseudo Code Format

Instructions are explained in the following pseudo-code format that resembles the GEN assembly instruction format.

```
[(pred)] opcode (exec_size) dst src0 [src1]
```

Square brackets "[ ]" indicate that a field is optional. Saturation modifiers and instruction options are omitted for simplicity.

General Macros and Definitions

INST_MIN_SIZE is defined as a constant of 8 bytes.

```
#define INST_MIN_SIZE  8  // Instruction minimum size in bytes (for the compact instruction format)
```

The floor function converts a floating point value to an integral floating point value. For a given floating point value, from its closest two integral float values, floor returns the one that is closer to negative infinity. For example, floor(1.3f) = 1.0f and floor(-1.3f) = -2.0f.

```
float floor(float g)
{
    return maximum(any integral float f: f <= g)
}
```

The Condition function takes the conditional signals {SN, ZR, OF, IN, NC} of result, generates a Boolean value according to a conditional evaluation controlled by the conditional modifier cmod, and returns the Boolean.

```
Bool Condition(result, cmod)
```

The ConditionNaN function takes the conditional signals {SN, ZR, OF, IN, NC, NS} of result, generates a Boolean value according to a conditional evaluation controlled by the conditional modifier cmod, and returns the Boolean. The only difference between Condition and ConditionNaN is that ConditionNaN uses the NS (NaN of the second source) signal.

```
Bool ConditionNaN(result, cmod)
```

The Jump function jumps the instruction sequence from the current instruction location by InstCount 8-byte units, where each 16-byte native instruction is two units and each 8-byte compact instruction is one unit. If InstCount is positive and greater than zero, is an unconditional jump forward. If InstCount is negative, is an unconditional jump backward. If InstCount is zero, IP stays on the current instruction in an infinite loop.

```
void Jump(int InstCount)
{
    IP = IP + (InstCount * INST_MIN_SIZE)
}
```

Evaluate Write Enable

The WrEn should be evaluated as below.

Note: MaskCtrl = NoMask (1) skips the check for PcIP[n] == ExIP before enabling a channel.

```
if ( MaskCtrl == 1 ) {
  for ( n = 0; n < exec_size; n++ ) {
    WrEn[n] = 1;
  }
}
else {
  for ( n = 0; n < exec_size; n++ ) {
    if ( PcIP[n] == ExIP ) {
      WrEn[n] = 1;
    }
    else {
      WrEn[n] = 0;
    }
  }
}

if ( PredCtrl != 0000b ) {
  for ( n = 0; n < exec_size; n++ ) {
    WrEn[n] = WrEn[n] & PMask[n];
  }
}

for ( n = exec_size; n < 32; n++ ) {
  WrEn[n] = 0;
}
```

EUISA Instructions

Symbol	Name	Source
add	**Addition**	EUISA
addc	**Addition with Carry**	EUISA
asr	**Arithmetic Shift Right**	EUISA
avg	**Average**	EUISA
bfe	**Bit Field Extract**	EUISA
bfi1	**Bit Field Insert 1**	EUISA
bfi2	**Bit Field Insert 2**	EUISA
bfrev	**Bit Field Reverse**	EUISA
brc	**Branch Converging**	EUISA
brd	**Branch Diverging**	EUISA
break	**Break**	EUISA
call	**Call**	EUISA
calla	**Call Absolute**	EUISA
cmp	**Compare**	EUISA
cmpn	**Compare NaN**	EUISA
csel	**Conditional Select**	EUISA
sendc	**Conditional Send Message**	EUISA
sendsc	**Conditional Split Send Message**	EUISA

cont	**Continue**	EUISA
cbit	**Count Bits Set**	EUISA
dp2	**Dot Product 2**	EUISA
dp3	**Dot Product 3**	EUISA
dp4	**Dot Product 4**	EUISA
dph	**Dot Product Homogeneous**	EUISA
else	**Else**	EUISA
endif	**End If**	EUISA
math	**Extended Math Function** • INV - Inverse • LOG – Logarithm • EXP - Exponent • SQRT - Square Root • RSQ - Reciprocal Square Root • POW - Power Function • SIN - SINE • COS - COSINE • INT DIV - Integer Divide • INVM/RSQRTM	EUISA
fbl	**Find First Bit from LSB Side**	EUISA
fbh	**Find First Bit from MSB Side**	EUISA
frc	**Fraction**	EUISA
goto	**Goto**	EUISA
halt	**Halt**	EUISA
if	**If**	EUISA
illegal	**Illegal**	EUISA
subb	**Integer Subtraction with Borrow**	EUISA
join	**Join**	EUISA
jmpi	**Jump Indexed**	EUISA
lzd	**Leading Zero Detection**	EUISA
line	**Line**	EUISA
lrp	**Linear Interpolation**	EUISA
and	**Logic And**	EUISA
not	**Logic Not**	EUISA
or	**Logic Or**	EUISA
xor	**Logic Xor**	EUISA

mov	**Move**	EUISA
movi	**Move Indexed**	EUISA
mul	**Multiply**	EUISA
mac	**Multiply Accumulate**	EUISA
mach	**Multiply Accumulate High**	EUISA
mad	**Multiply Add**	EUISA
madm	**Multiply Add for Macro**	EUISA
nop	**No Operation**	EUISA
pln	**Plane**	EUISA
ret	**Return**	EUISA
rndd rnde rndu rndz	**Round Instructions** ▪ **Round Down** ▪ **Round to Nearest or Even** ▪ **Round Up** ▪ **Round to Zero**	EUISA
smov	**Scattered Move**	EUISA
sel	**Select**	EUISA
send	**Send Message**	EUISA
shl	**Shift Left**	EUISA
shr	**Shift Right**	EUISA
sends	**Split Send Message**	EUISA
sad	**Sum of Absolute Difference 2**	EUISA
sada2	**Sum of Absolute Difference Accumulate 2**	EUISA
wait	**Wait Notification**	EUISA
while	**While**	EUISA

Round Instructions

rndd - Round Down

rndu - Round Up

rnde - Round to Nearest or Even

rndz - Round to Zero

rndd – Round Down

Description:

The *rndd* instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of src0 and storing the rounded integral float results in dst. This is commonly referred to as the floor() function.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round Down in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	-finite	^	–0	+0	+0	**	+inf	NaN
Notes:									
^	Note: {-1, -0} depending on the Single Precision Denorm Mode.								
**	Result may be {+finite, +0}.								

Floating-Point Round Down in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	-finite	–0	–0	+0	+0	**	+fmax	
Notes:									
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rnde – Round to Nearest or Even

Description:

The *rnde* instruction takes component-wise floating point round-to-even operation of src0 with results in two pieces – a downward rounded integral float results stored in dst and the round-to-even increments stored in the rounding increment bits. The round-to-even increment must be added to the results in dst to create the final round-to-even values to emulate the round-to-even operation, commonly known as the round() function. The final results are the one of the two integral float values that is nearer to the input values. If the neither possibility is nearer, the even alternative is chosen.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round to Nearest or Even in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	+0	**	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								

Floating-Point Round to Nearest or Even in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	**	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rndu – Round Up

Description:

The *rndu* instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of src0, commonly known as the ceiling() function.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round Up in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	^	+finite	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
^	Note: {+1, +0} depending on the Single Precision Denorm Mode.								

Floating-Point Round Up in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	+finite	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rndz – Round to Zero

Description:

The *rndz* instruction takes component-wise floating point round-to-zero operation of src0 with results in two pieces – a downward rounded integral float results stored in dst and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in dst to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the truncate() function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Floating-Point Round to Zero in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	+0	**	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								

Floating-Point Round to Zero in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	**	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

math – Extended Math Function

Function
math - Extended Math Function

Description:

The *math* instruction performs extended math function on the components in src0, or src0 and src1, and write the output to the channels of dst. The type of extended math function are based on the FC[3:0] encoding in the table below.

Function Control[3:0]	Function Description
0h	Reserved
1h	INV (reciprocal)
2h	LOG
3h	EXP
4h	SQRT
5h	RSQ
6h	SIN
7h	COS
8h	Reserved
9h	FDIV
Ah	POW
Ch	INT DIV – return quotient only
Dh	INT DIV – return remainder
Eh	INVM
Fh	RSQRTM

INV - Inverse

Precision:1 ULP

Src->	+inf	+0 / +Denorm	- 0 / -Denorm	-inf	NaN
Dest – IEEE mode	+0	+inf	-inf	-0	NaN
Dest – ALT mode		+fmax	-fmax		NaN
Src->	+inf	+0	- 0	-inf	NaN
Dest – IEEE mode	+0	+inf	-inf	-0	NaN

LOG – Logarithm

Precision:

DirectX 10 and below

If src0 is [0.5..2], **absolute error** must be no more than 2-21. If src0 is (0..0.5) or (2..+INF], **relative error** must be no more than 2-21

Note: In ALT mode log is computed as Log_2 (abs (src0))

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	**+inf**	**-inf**	**-inf**	**NaN**	**NaN**	**NaN**
Dest – ALT mode		**-fmax**	**-fmax**		**+F**	**NaN**
Src->	+inf	+0	- 0	-inf	-F	NaN
Dest – IEEE mode	**+inf**	**-inf**	**-inf**	**NaN**	**NaN**	**NaN**

EXP - Exponent

Precision:

DirectX 10 Relative error <= 3 ULP

GPGPU – Relative error <= 4 ULP

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	**+inf**	**1**	**1**	**0**	**+F**	**NaN**
Dest – ALT mode		**1**	**1**		**+F**	**NaN**
Src->	+inf	+0	- 0	-inf	-F	NaN
Dest – IEEE mode	**+inf**	**1**	**1**	**0**	**+F**	**NaN**

SQRT - Square Root

Precision:

DirectX 10 (and below) Relative error <= 1 ULP

GPGPU – Relative error <= 4 ULP

Notes:In ALT mode SQRT is computed as SQRT(abs (src0))

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	**+inf**	**0**	**-0**	**NaN**	**NaN**	**NaN**
Dest – ALT mode		**0**	**0**		**+F**	**NaN**
Src->	+inf	**+0**	**-0**	-inf	**-F**	**NaN**
Dest – IEEE mode	+inf	**0**	**-0**	NaN	**NaN**	**NaN**

RSQ - Reciprocal Square Root

Precision:

DirectX 10 and below Relative error <= 3 ULP

GPGPU – Relative error <= 4 ULP

Notes:In ALT mode RSQ is computed as RSQ(abs (src0))

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	**+0**	**+inf**	**-inf**	**NaN**	**NaN**	**NaN**
Dest – ALT mode		**+fmax**	**+fmax**		**+F**	**NaN**
Src->	+inf	+0	-0	-inf	-F	NaN
Dest – IEEE mode	**+0**	**+inf**	**-inf**	**NaN**	**NaN**	**NaN**

POW - Power Function

Precision:

DirectX 10+ do not have precision requirements for POW.

Half_pow precision for GPGPU; i.e., 8192 ULPs with following exception

POWR, POWN is not supported in hardware.

IEEE Mode:

Src0->								
Src1	abs(F > 1)	abs(F < 1)	abs(+F = = 1)	+inf	+0 / +Denorm	-Denorm / -0	-inf	NaN
+inf	+inf	0	NaN	+inf	0	0	+inf	NaN
+0 / Denorm	1	1	1	NaN	NaN	NaN	NaN	NaN
-0 / Denorm	1	1	1	NaN	NaN	NaN	NaN	NaN
-inf	0	+inf	NaN	0	+inf	+inf	0	NaN
-F	+F	+F	+F	0	+inf	+inf	0	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+F	+F			+inf	0	0	NaN	NaN

IEEE Mode:

Src0->								
Src1	abs(F > 1)	abs(F < 1)	abs(+F = = 1)	+inf	+0	-0	-inf	NaN
+inf	+inf	0	NaN	+inf	0	0	+inf	NaN
+0	1	1	1	NaN	NaN	NaN	NaN	NaN
-0	1	1	1	NaN	NaN	NaN	NaN	NaN
-inf	0	+inf	NaN	0	+inf	+inf	0	NaN
-F	+F	+F	+F	0	+inf	+inf	0	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
+F	+F			+inf	0	0	NaN	NaN

ALT Mode:

Src0->							
Src1	+F	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
+inf							
+0 / Denorm	1		1	1		1	NaN
-0 / Denorm	1		1	1		1	NaN
-inf							
-F	+F		+fmax	+fmax		+F	NaN
NaN			NaN	NaN		NaN	NaN
+F	+F		0	0		+F	NaN

SIN - SINE

Precision:

DirectX 10 and below Absolute error <= 0.0008 for the range of +/- 32767* pi

Outside of the above range the function will remain periodic, producing values between -1 and 1. However, the period of SIN is determined by the internal representation of Pi, meaning that as the magnitude of input increases the absolute error will, in general, also increase.

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	NaN	+0	-0	NaN	-1 to 1	NaN
Dest – ALT mode		+0	-0		-1 to 1	NaN
Src->	+inf	+0	-0	-inf	-F	NaN
Dest – IEEE mode	NaN	+0	-0	NaN	-1 to 1	NaN

COS - COSINE

Precision:

DirectX 10 and below Absolute error <= 0.0008 for the range of +/- 32767* pi

Outside of the above range the function will remain periodic, producing values between -1 and 1. However, the period of COS is determined by the internal representation of Pi, meaning that as the magnitude of input increases the absolute error will, in general, also increase.

Src->	+inf	+0 / +Denorm	-0 / -Denorm	-inf	-F	NaN
Dest – IEEE mode	NaN	+0	-0	NaN	-1 to 1	NaN
Dest – ALT mode		+1	+1		-1 to 1	NaN
Src->	+inf	+0	-0	-inf	-F	NaN
Dest – IEEE mode	NaN	+0	-0	NaN	-1 to 1	NaN

INT DIV - Integer Divide

Precision:32-bit integer

For signed inputs, INT DIV behavior is illustrated by the table below:

Inputs:	**Numerator**	**+**	**+**	**-**	**-**
	Denominator	**+**	**-**	**+**	**-**
Outputs:	**Quotient**	**+**	**-**	**-**	**+**
	Remainder	**+**	**+**	**-**	**-**

INVM/RSQRTM

These are special opcodes. These two special opcodes INVM and RSQRTM provide the initial approximation for a macro that finally produces an IEEE compliant FDIV and SQRT respectively. They perform the same operation as INV and RSQRT. However, the outputs of these opcodes have higher precision, i.e., 34 bits for single precision and 66 bits for double precision. 32 bits and 64 bits, for single and double precision respectively are stored in the GRF and the remaining bits are stored in the special accumulators acc2-acc9. These operations provide the initial approximation for inverse and sqrt operations. The final IEEE compliant values are derived through a series of special fma operations. The pseudo code for these macro operations is defined below.

Programming Notes: The GRF registers and accumulators used in the macros below are merely examples. There are no restrictions on using these in any specific order as long as the GRF register and the accumulator are paired correctly in writes and reads. Since early out handling requires flow control to channels that do not require compute through the macro, Single Program Flow must never be set for a macro operation.

Macro for Single Precision IEEE Compliant fdiv

[*Note: Constants are in quotes.*]

```
Set Rounding Mode in CR to RNE
GRF are initialized: r2 = 0, r6 = a, r4 = b, r5 = 1
The default data type for the macro is :f

math.eo.f0.0 (8) r8.acc2 r6.noacc r4.noacc 0xE
(-f0.0) if
madm (8) r9.acc3 r2.noacc r6.noacc r8.acc2    // Step(1), q0=a*y0
madm (8) r10.acc4 r5.noacc -r4.noacc r8.acc2  // Step(2), e0=(1-b*y0)
madm (8) r1.acc5 r8.acc2 r10.acc4 r8.acc2     // Step(3), y1=y0+e0*y0
madm (8) r11.acc6 r6.noacc -r4.noacc r9.acc3  // Step(4), r0=a-b*q0
madm (8) r9.acc7 r9.acc3 r11.acc6 r1.acc5     // Step(5), q1=q0+r0*y1
madm (8) r6.acc8 r6.noacc -r4.noacc r9.acc7   // Step(6), r1=a-b*q1

Change Rounding Mode in CR if required
Implicit Accumulator for destination is NULL

madm (8) r8.noacc r9.acc7 r6.acc8 r1.acc5      // Step(7), q=q1+r1*y1
endif
```

Macro for Double Precision IEEE Compliant fdiv

```
Set Rounding Mode in CR to RNE
GRF are initialized: r0 = 0, r6 = a, r7 = b, r1 = 1
The default data type for the macro is :df

math.eo.f0.0 (4) r8.acc2 r6.noacc r7.noacc 0xE
```

```
(-f0.0) if
madm (4) r9.acc3 r0.noacc r6.noacc r8.acc2    // Step(1), q0=a*y0
madm (4) r10.acc4 r1.noacc -r7.noacc r8.acc2  // Step(2), e0=(1-b*y0)
madm (4) r11.acc5 r6.noacc -r7.noacc r9.acc3  // Step(3), r0=a-b*q0
madm (4) r12.acc6 r8.acc2 r10.acc4 r8.acc2    // Step(4), y1=y0+e0*y0
madm (4) r13.acc7 r1.noacc -r7.noacc r12.acc6 // Step(5), e1=(1-b*y1)
madm (4) r8.acc8 r8.acc2 r10.acc4 r12.acc6    // Step(6), y2=y0+e0*y1
madm (4) r9.acc9 r9.acc3 r11.acc5 r12.acc6    // Step(7), q1=q0+r0*y1
madm (4) r12.acc2 r12.acc6 r8.acc8 r13.acc7   // Step(8), y3=y1+e1*y2
madm (4) r11.acc3 r6.noacc -r7.noacc r9.acc9  // Step(9), r1=a-b*q1

Change Rounding Mode in CR if required
Implicit Accumulator for destination is NULL

madm (4) r8.noacc r9.acc9 r11.acc3 r12.acc2   // Step(10), q=q1+r1*y3
endif
```

Macro for Single Precision IEEE Compliant sqrt

```
Set Rounding Mode in CR to RNE
GRF are initialized: r0 = 0, r8 = 1/2, r6 = a
The default datatype for the macro is :f

math.eo.f0.0 (8) r7.acc2 r6.noacc null 0xF
(-f0.0) if
madm (8) r9.acc3 r0.noacc r8.noacc r7.acc2    // Step(1), H0=1/2*y0
madm (8) r11.acc4 r0.noacc r6.noacc r7.acc2   // Step(2), S0=a*y0
madm (8) r10.acc5 r8.noacc -r11.acc4 r9.acc3  // Step(3), d0=1/2-S0*H0
madm (8) r9.acc6 r9.acc3 r10.acc5 r9.acc3     // Step(4), H1=H0+d0*H0
madm (8) r7.acc7 r11.acc4 r10.acc5 r11.acc4   // Step(5), S1=S0+d0*S0
madm (8) r10.acc8 r6.noacc -r7.acc7 r7.acc7   // Step(6), e0=a-S1*S1

Change Rounding Mode in CR if required
Implicit Accumulator for destination is NULL

madm (8) r7.noacc r7.acc7 r9.acc6 r10.acc8    // Step(7), S=S1+e0*H1
endif
```

Macro for Double Precision IEEE Compliant sqrt

```
Set Rounding Mode in CR to RNE
GRF are initialized: r0 = 0, r8 = 1/2, r6 = a, r1 = 1
The default datatype for the macro is :df

math.eo.f0.0 (4) r7.acc2 r6.noacc null 0xF
(-f0.0) if
madm (4) r9.acc3 r0.noacc r8.noacc r7.acc2    // Step(1), H0=1/2*y0
madm (4) r11.acc4 r0.noacc r6.noacc r7.acc2   // Step(2), S0=a*y0
```

```
madm (4) r10.acc5 r8.noacc -r11.acc4 r9.acc3  // Step(3), d=1/2-S0*H0
madm (4) r8.acc6 r1.noacc r8.noacc r1.noacc   // Step(0'') 3/2=1+1/2
madm (4) r8.acc7 r1.noacc r8.acc6 r10.acc5    // Step(4), e=1+3/2*d
madm (4) r7.acc8 r0.noacc r10.acc5 r11.acc4   // Step(5), T0=d*S0
madm (4) r10.acc9 r0.noacc r10.acc5 r9.acc3   // Step(6), G0=d*H0
madm (4) r7.acc8 r11.acc4 r8.acc7 r7.acc8     // Step(7), S1=S0+e*T0
madm (4) r8.acc7 r9.acc3 r8.acc7 r10.acc9     // Step(8), H1=H0+e*G0
madm (4) r9.acc3 r6.noacc -r7.acc8 r7.acc8    // Step(9), d1=a-S1*S1

Change Rounding Mode in CR if required
Implicit Accumulator for destination is NULL

madm (4) r7.noacc r7.acc8 r9.acc3 r8.acc7  // Step(10), S=S1+d1*H1
endif
```

Restrictions for the Use of Macros

3. Macro Operations need to be used as special subroutines. Interleaving other operations is not recommended.
4. Single precision operations must be SIMD8 and double precision operations must be SIMD4.
5. The macro sequences should be used with the Rounding Mode in CR set to RNE. The Rounding Mode may be changed only for the last *madm* of the macro depending on the requirement.

Workaround
When both srcs are NAN, FDIV produces denorminator NAN as output.

Send Message

send - Send Message

The following information describes the send message.

Send Message

Opcode	Instruction	Description
49 (0x31)	send <dest> <src> <ex_desc> <desc>	Send a message stored in GRF starting at <src> to a shared function identified by <ex_desc> along with control from <desc> with a GRF writeback location at <dest>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Type
Y					[FLT] [INT]

Format:

```
[(pred)] send (exec_size) <dest> <src> <ex_desc> <desc>
```

Syntax:

Syntax
[(pred)] send (exec_size) reg greg imm32 reg32a [(pred)] send (exec_size) reg greg imm32 imm32

Pseudocode:

```
Evaluate(WrEn);
<MsgChEnable> = WrEn;
<SourceReg> = <src>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg>, <ex_dest>,
<desc>);
```

Description:

The *send* instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The *send* instruction adds an entry to the EU's message request queue. The request message is stored in a block of contiguous GRF registers. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src> is the lead GRF register for request. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend mesage descriptor field <ex_desc> contains the target function ID. WrEn is forwarded to the target function in the message sideband.

The extended message descriptor field <ex_desc> also contains the extended function control field to be sent to the Target Shared Function over message sideband.

The *send* instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.

Message descriptor field <desc> can be a 32-bit immediate, *imm32*, or a 32-bit scalar register, <reg32a>. GEN restricts that the 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0<0;1,0>:ud. When <desc> is a register operand, only the lower 29 bits of <reg32a> are used.

Syntax
<ex_desc> is a 32-bit immediate, imm32. The lower 4bits of the <ex_desc> specifies the SFID for the message. The bit5 of the extended message descriptor, the EOT field, always comes from bit 127 of the instruction word. A thread must terminate with a send instruction with EOT turned on. The higher 16bits, bit31:16 specify the 16bit extended function control field. Interpretation of the extended function control signals is subject to the target external function.

<src> is a 256-bit aligned GRF register. It serves as the leading GRF register of the request.

The source dependency control, {NoSrcDepSet} is used to control the setting of source dependency for the source.

<dest> serves for two purposes: to provide the leading GRF register location for the response message if present, and to provide parameters to form the channel enable sideband signals.

<dest> signals whether there is a response to the message request. It can be either a null register, a direct-addressed GRF register or a register-indirect GRF register. Otherwise, hardware behavior is undefined.

If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null.

If <dest> is a GRF register, the register number is forwarded to the shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off.

The subregister number, horizontal stride, destination mask and type fields of <dest> are always valid and are used in part to generate on the WrEn. This is true even if <dest> is a null register (this is an exception for null as for most cases these fields are ignored by hardware).

The 16-bit channel enables of the message sideband are formed based on the WrEn. Interpretation of the channel enable sideband signals is subject to the target external function. In general for a 'send' instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases.

Thread managed memory coherency: A special usage of using non-null <dest> is to support write-commit signaling for memory write service by the Data Port Write unit. If <post_dest> is not null for a memory write request, the Data Port along with the Data Cache or Render Cache will wait until all the posted writes for the request have reached the coherent domain before sending back to the requesting thread an empty message to <dest> register. A memory write reaching the coherent domain, also referred to as reaching the global observable state, means that subsequent read to the same memory location, no matter which thread issues the read, must return the data of the write.

NoDDClr and NoDDChk must not be used for send instruction.

r127 must not be used for return address when there is a src and dest overlap in send instruction.

Message Descriptor Definition

Description
Reserved : MBZ
Data Format. This field specifies the width of data read from sampler or written to render target. Format = U1 0 – Single Precision (32b) 1 – Half Precision (16b)

Reserved : MBZ
Message Length. This field specifies the number of 256-bit GRF registers starting from <src> to be sent out on the request message payload. Valid value ranges from 1 to 15. A value of 0 is considered erroneous. Format = U4 Range = [1,15]
Response Length. This field indicates the number of 256-bit registers expected in the message response. The valid value ranges from 0 to 16. A value 0 indicates that the request message does not expect any response. The largest response supported is 16 GRF registers. Format = U5 Range = [0,16]
Header Present. If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Format = Enable
Function Control This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.

Extended Message Descriptor Definition

Bit	Description
31:16	Extended Function Control. This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
15:6	reserved
5	**End Of Thread** This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
4	reserved
3:0	**Target Function ID** This field indicates the function unit for which the message is intended. ***Refer to "GPU Overview" document for the mapping of Shared Function IDs***

Restrictions:

Software must obey the following rules in signaling the end of thread using the *send* instruction:

- The posted destination operand must be null.
- No acknowledgement is allowed for the *send* instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource.
- A thread must terminate with a *send* instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a *send* instruction with message to the following shared functions: Sampler unit, NULL function. For example, a thread may terminate with a URB write message or a render cache write message.
- A root thread originated from the media (generic) pipeline must terminate with a *send* instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed description.
- The *send* instruction can not update accumulator registers.
- Saturate is not supported for *send* instruction.
- ThreadCtrl are not supported for *send* instruction.
- The *send* with EOT should use register space R112-R127 for <src>. This is to enable loading of a new thread into the same slot while the message with EOT for current thread is pending dispatch.
- SEND should not use 64-bit datatype for src or dest register.
- When context save and restore of a thread is required, the registers r0-r4 must not be used as *dest or src* registers. This is required to provide a grf register space to save sr1.0 by the system interrupt routine in the event of a mid thread pre-emption.

Extended Message Descriptor Definition

Bits	Description
5	**End Of Thread**This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
4	reserved
3:0	**Target Function ID** This field indicates the function unit for which the message is intended. Refer to volume GPU Overview for GPE Function IDs.

EUISA Structures

Name	Source
AddrSubRegNum	Eulsa
DstRegNum	Eulsa
DstSubRegNum	Eulsa

Name	Source
EU_INSTRUCTION_BASIC_ONE_SRC	Eulsa
EU_INSTRUCTION_BASIC_THREE_SRC	Eulsa
EU_INSTRUCTION_BASIC_TWO_SRC	Eulsa
EU_INSTRUCTION_BRANCH_CONDITIONAL	Eulsa
EU_INSTRUCTION_BRANCH_ONE_SRC	Eulsa
EU_INSTRUCTION_BRANCH_TWO_SRC	Eulsa
EU_INSTRUCTION_COMPACT_THREE_SRC	Eulsa
EU_INSTRUCTION_COMPACT_TWO_SRC	Eulsa
EU_INSTRUCTION_CONTROLS	Eulsa
EU_INSTRUCTION_CONTROLS_A	Eulsa
EU_INSTRUCTION_CONTROLS_B	Eulsa
EU_INSTRUCTION_HEADER	Eulsa
EU_INSTRUCTION_ILLEGAL	Eulsa
EU_INSTRUCTION_MATH	Eulsa
EU_INSTRUCTION_NOP	Eulsa
EU_INSTRUCTION_OPERAND_CONTROLS	Eulsa
EU_INSTRUCTION_OPERAND_DST_ALIGN1	Eulsa
EU_INSTRUCTION_OPERAND_DST_ALIGN16	Eulsa
EU_INSTRUCTION_OPERAND_SEND_MSG	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC	Eulsa
EU_INSTRUCTION_SEND	Eulsa
EU_INSTRUCTION_SENDS	Eulsa
EU_INSTRUCTION_SOURCES_IMM32	Eulsa
EU_INSTRUCTION_SOURCES_REG	Eulsa
EU_INSTRUCTION_SOURCES_REG_IMM	Eulsa
EU_INSTRUCTION_SOURCES_REG_REG	Eulsa
ExtMsgDescpt	Eulsa
FunctionControl	Eulsa
MsgDescpt31	Eulsa
SrcRegNum	Eulsa
SrcSubRegNum	Eulsa

EUISA Enumerations

Name	Source
AddrMode	Eulsa
ChanEn	Eulsa
ChanSel	Eulsa
CondModifier	Eulsa
DepCtrl	Eulsa
DstType	Eulsa
EU_OPCODE	Eulsa
ExecSize	Eulsa
FC	Eulsa
HorzStride	Eulsa
PredCtrl	Eulsa
QtrCtrl	Eulsa
RegFile	Eulsa
RepCtrl	Eulsa
SFID	Eulsa
SrcImmType	Eulsa
SrcIndex	Eulsa
SrcMod	Eulsa
SrcType	Eulsa
ThreadCtrl	Eulsa
VertStride	Eulsa
Width	Eulsa

EU Programming Guide

Assembler Pragmas

Declarations

A register or a register region can be declared as a symbol using the following form

.declare <symbol> **Base**=RegFile RegBase {.SubRegBase} **ElementSize**=ElementSize {**SrcRegion**=DefaultSrcRegion} {**DstRegion**=DefaultDstRegion} {**Type**=DefaultType}

The register file, the base of the register origin and the element size (in unit of bytes) are the mandatory parameters for a declared register region. Optionally, the base of the sub-register address, the default source region, the default destination region and the default type can be provided in the declaration for the symbol.

For immediate register addressing mode, the declared symbol can be used in the following Cartesian form

<symbol>(RegOff, SubRegOff)<=RegNum = *RegBase*+ **RegOff***;* SubRegNum = *SubRegBase*+ **SubRegOff**

or in the following simplified row-aligned form

<symbol>(RegOff)<=RegNum = *RegBase*+ **RegOff;** SubRegNum = *SubRegBase*

For register-indirect-register-addressing mode, the declared symbol can be used to provide immediate address term in the following Cartesian form

<symbol>[IdxReg, RegOff, SubRegOff]<= RegNum (byte-aligned) = [**IdxReg]**+(*RegBase*+ **RegOff**)*32 + (*SubRegBase* + **SubRegOff**)**ElementSize*

or in the following simplified row-aligned form

<symbol>[IdxReg, RegOff]<= RegNum (byte-aligned) = [**IdxReg]**+(*RegBase*+ **RegOff**)*32

or in the form without the immediate address term

<symbol>[IdxReg]<= RegNum (byte-aligned) = [**IdxReg]**+ *RegBase*

Defaults and Defines

The default execution size is set according to the destination register type as the following

Destination Register Type	Default Execution Size
UB | B	(16)
UW | W	(16)
F | UD | D	(8)

The default execution size can be overwritten globally for all instructions using

.**default_execution_size***(Execution_Size)*

or be set according the **destination** register type using

.**default_execution_size**_*Type(Execution_Size)*

The default register type can be set for all register files using

.**default_register_type***Type*

or be set per register file using

.**default_register_type**_*RegFileType*

The default **source** register region for all symbols can be set using

.**default_source_register_region***<VirtStride; Width, HorzStride>*

or be set per register type using

.**default_source_register_region**_*type<VirtStride; Width, HorzStride>*

The default **destination** register region for all symbols can be set using

.**default_destination_register_region***< HorzStride>*

or be set per register type using

.**default_destination_register_region**_*type< HorzStride>*

Finally, the precompiler supports the string replacement statement of .define in the following form

.**define**<symbol>Expression

Programming Note	
Context:	Defaults and Defines
• **.declare** does not support nesting. In other words, each symbol in .declare must be self defined. This would allow the pre-processor to expand all symbols in one pass. • **.define** does support nesting. Only string substitution is supported (currently). • White space within square, angle and round brackets are allowed for easy source code alignment.	

Example Pragma Usages

Example: Declaration for 8x4=32-Byte Regions:

The following symbol Block can be used to address any 8x4 byte region within the Cartisian system of a 16x8 byte GRF register area starting from r0.

Declaration

```
// 32x4 Byte Array.declare BlockBase=r0 ElementSize=1 Region=<32;8,1>Type=b
```

Fully-Expressed Instr

```
mov(32)?:br0.16<32;8,1>:b// r0 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r1
xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r2 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r3
xxxxxxxxxxxxxxxxoooooooooxxxxxxxx
```

Short-handed Instr

Mov?:bBlock(0,16)// (0,16): RegNum=0, SubRegNum=16

Example: Declaration for 8x1 Float Regions:

The following symbol *Trans* can be used to address any 8x1 float region within the Cartisian system of a 8x4 float GRF register area starting from r5.

Declaration

```
// 8x4 float Array starting at r5.declare Trans Base=r5 ElementSize=4 Region=<0;8,1> Type=f
```

Fully-Expressed Instr

```
mov(8)?:fr6.0<0;8,1>:f// 2nd 16x1 Row of Trans. Matrix // r5 FFFFFFFF// r6 OOOOOOOO// r7
FFFFFFFF// r8 FFFFFFFF
```

Short-handed Instr

mov?:fTrans(1) // RegNum = 5+1 = 6

Example: Declaration for 8x1 Float Regions with 1x1 Indirect Addressing:

Trans region defined (same as in the previous example) is used in conjunction with the address register.

Declaration

```
//8x4 float data array and 16x1 word address array.declare TransBase=r5 ElementSize=4
Region=<0;8,1> Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,224]<0;8,1>:f

Short-handed Instr

mov?:fTrans[a0.0,2] // [a0.0 + 5*32 + 2*32]

Example: Declaration with VxH Indirect Addressing:

The VxH register-indirect-register-addressing for Trans can be provided in the following short-hand form

Declaration

```
//8x4 float data array and word indices.declare TransBase=r5 ElementSize=4 Region=<0;8,1>
Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,224]<1,0>:f

Short-handed Instr

mov?:fTrans[a0.0,2]<1,0> // [a0.0+224] [a0.1+224] ... [a0.7+224]

Example: Declaration with Vx1 Indirect Addressing:

```
As width (4) is smaller than the execution region size (8), multiple indexed registers are
used.
```

Declaration

```
//8x4 float data array and word address array.declare TransBase=r5 ElementSize=4
Region=<0;8,1> Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,244]<4,1>:f

Short-handed Instr

```
mov?:fTrans[a0.0,2]<4,1>// [a0.0+224] [a0.1+224]
```

Assembly Programming Guideline

The following program skeleton illustrates the basic structure of a typical assembly program.

```
                          // single line comment

 /*          block comment
 */

 <preproc_directive>             // macros, include, etc.  Are global – handled by the pre-
processor
 <preproc_directive>             // applies to all code that follows in sequence

 // ------------ some kernel
 .kernel <kernel_name_string>    // [REQUIRED]

 // ------- Register requirements --------
 .reg_count_total    <uint>      // [REQUIRED] a more direct way to specify the parameters
required
 .reg_count_payload  <uint>      // [REQUIRED] rather than indirectly adding the
                          // the payload and temps together to get the total (as is the case
now)
                          // Note: no more "reg-count-temp"

 // -------------- Defaults ---------------
 <default…>               // these should be specified per-kernel and have only kernel-scope
 <default…>               // Same defaults as those already defined in the ISA doc, but just
 <default…>               // moved within the kernel to make each kernel completely self-
sufficient
                          // and not impacted defaults of earlier kernels

 // --------- Memory Requirements ---------
 // [optional] memory block info (just a placeholder for now...)

 <MBDa>                   // memory block descriptor a (TBD)
 <MBDb>                   // memory block descriptor b (TBD)
 <MBDc>                   // memory block descriptor c (TBD)
 <MBDd>                   // memory block descriptor d (TBD)

 // ---------------- Code  ----------------
 .code                           // [REQUIRED]
   <instruction>
   <instruction>
   <instruction>
 <LabelLine>                     // labels are code-block scope
   <instruction>
```

```
  <instruction>
.end_code                  // [REQUIRED]

.end_kernel                        // [REQUIRED]

// --------- next kernel -------------

// --------- next kernel -------------

// ...
```

Usage Examples

This topic is currently under development.

Vector Immediate

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. An immediate vector is denoted by type v as *imm32:v*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Each 4-bit subfield contains a signed integer value. Therefore each 4-bit subfield has a range of [-8, +7]. This is depicted in the following figure.

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
V7	V6	V5	V4	V3	V2	V1	V0

Supporting DirectX 10 Pixel Shader Indexing

When a DirectX 10 Pixel Shader program is converted to run on GEN in channel-serial mode at 16 pixels in parallel, the per-pixel index must be translated into 16 indices with per channel offset. The creation of the per-channel offset can be achieved using the vector immediate.

Consider a generic DirectX 10 Pixel Shader instruction in the form of

opr4r[ind]r2

and assume that r0-r1 contain the 16 indices packed every other words, and r2-r3 contains source 1 and r4-r5 contain the destination. This instruction can be converted into the following GEN instructions. The corresponding operations are illustrated in *Supporting DirectX 10 Pixel Shader Indexing*.

mov (16) r11.0<1>:w 0x01234567:v// assigning a ramp vector, repeated once

mul (16)acc0:wr11.0<0;16,1>:w4:w// expand ramp range to 4 bytes per step

mac (16)r10.0<1>:wr0.0<16;8,2>:w32:w// r10 = index*32 + 0|4|...|28|0|4...|28

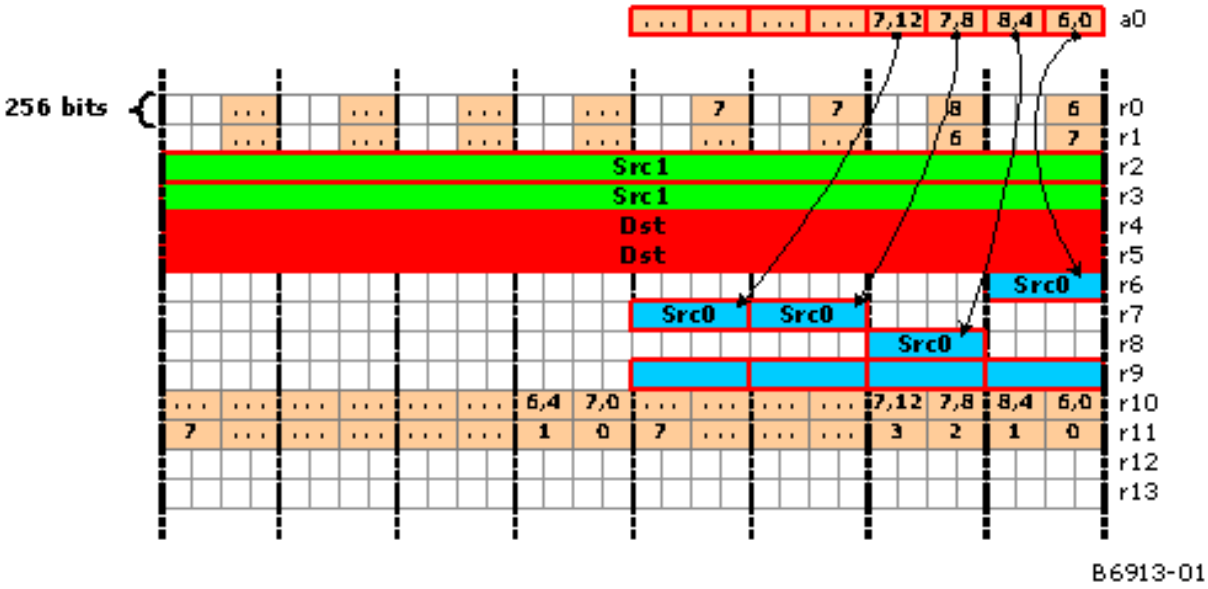
mov (8)a0.0<1>:wr10.0<0;8,1>:w

op (8)r4.0<1>:fr[a0.0]<1,0>:fr2.0<0;8,1>:w// Operate on the first half

mov (8)a0.0<1>:wr10.8<0;8,1>:w// Index values are off by a reg (32b)

op (8)r5.0<1>:fr[a0.0+32]<1,0>:fr3.0<0;8,1>:w// Operate on the second half.

Pixel Shader example using vector immediate.



Without vector immediate support, such translation has to either use a long sequence of scalar instructions which is very inefficient or use a constant load which requires additional constant to be managed in memory.

Supporting OpenGL Vertex Shader Instruction SWZ

When an OpenGL Vertex Shader program is converted to run on GEN in Vertex Pair, i.e. two 4-wide vectors in parallel, the special OpenGL Shader instruction SWZ (Swizzle) needs to be emulated. OpenGL SWZ instruction uses an extended swizzle control field that, in addition to the 4-wide full swizzle control, also includes constant 0 and 1 replacement as well as per channel sign reversal. The later two are not supported by the GEN native instruction. The vector immediate can significantly reduce the overhead of emulating such OpenGL instruction.

Consider an OpenGL Shader instruction in the form of

```
SWZ r1 r0.0-zx-1 // Expected results: r1.x = 0; r1.y = -r0.z; r1.z = r0.x; r1.w = -1
```

It can be emulated by the following three GEN instructions.

```
mul (8) r1.0<1>:f r0.xzxz 0x1F111F11:v // Constant vector of (1 -1 1 1 1 -1 1 1)
mov (1)f0.0 8b'10011001                // Set flag & masked out channels y and z
(f0.0)mov(8) r1.0<1>:f 0x000F000F:v    // Constant vector of (0 0 0 -1 0 0 0 -1)
```

In case that only 0, 1, -1 channel replacement is used and there is no signed swizzle, it may be emulated in two GEN instructions. This is illustrated by the following example:

OpenGL:

```
SWZ r1 r0.0zx-1 // Expected results: r1.x = 0; r1.y = r0.z; r1.z = r0.x; r1.w = -1
```

GEN:

```
mov (1)f0.0 8b'01100110                     // Set flag and masked out channels x and w
(f0.0)sel (8) r1.0<1>:f r0.yzxy 0x000F000F:v // Constant vector of (0 0 0 -1 0 0 0 -1)
```

Destination Mask for DP4 and Destination Dependency Control

The following example demonstrates the use of destination mask mode of floating point dot-product instruction as well as the use of destination dependency control to improve performance (i.e., avoiding unnecessary thread switch due to possible false dependencies).

Consider a generic DirectX 10 Vertex Shader macro of matrix-vector product that is implemented on GEN in the pair of 4-component vector mode. The DirectX 10 equivalent Shader instructions are as the following.

dp4 r5.x r0 r4

dp4 r5.y r1 r4

dp4 r5.z r2 r4

dp4 r5.w r3 r4

With destination dependency control, the GEN instructions are as the following. The first instruction in the sequence checks for the destination dependency, but does not clear the dependency bit. The subsequent two instructions would do neither of them. The last instruction avoids checking the destination dependency, but at completion, it clears the destination scoreboard. It ensures that the content of the destination register is coherent, if any of the following instructions uses the same register as source.

dp4 (8) r5.0<1>.x:f r0.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr}

dp4 (8) r5.0<1>.y:f r1.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr, NoDDCChk}

dp4 (8) r5.0<1>.z:f r2.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr, NoDDCChk}

dp4 (8) r5.0<1>.w:f r3.0<4;4,1>:f r4.0<4;4,1>:f {NoDDChk}

Just as a comparison, IF GEN DP4 implies reduction at the destination; additional shifted moves are required to achieve the same results. The corresponding codes are as the following. The lower performance due to the additional three move instruction as well as added back-to-back dependencies shows that why we choose to implement the destination channel replication for floating point DP4.

dp4 (8) r5.0<1>.y:f r1.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.1<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.z:f r2.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.2<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.w:f r3.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.3<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.x:f r0.0<4;4,1>:f r4.0<4;4,1>:f

Null Register as the Destination

Null register can be used as the destination for most of the instructions. Here are some example usages.

- Null as destination for regular ALU instructions: As all ALU instructions can be configured to update the flag registers using the conditional modifiers, it is not necessary to have a destination register if the programmer only cares about the conditionals of the operation. In that case, a null in the destination operand field saves register space as well as one less dependency checking.
- Null as the destination for SEND/STOR instructions: for the send instruction that only send messages out to an external unit and does not require any return data or feedback, a null in the destination register field signifies the case.
    - One extension of such case is that even though the operation does not have any return values, a return phase with no payload but simply updating the scoreboard flag for a non-null register can provide a signaling mechanism between the thread and the target external unit. One application of this usage is to allow software to manage the coherency of shared memory resources such like the many caches in the system (particularly, valuable for read/write caches).

Use of LINE Instruction

LINE instruction is specifically designed to speed up floating point vector/matrix computation when a program operates in channel serial.

The following example demonstrates how to use LINE instruction to compute Line Equations for DirectX 10 Pixel Shader. In this example, 2 sets of (Cx#, Cy#, Don't Care, C0#) 4-tuple coefficient vectors are stored in registers R1.

R1: Cx0 Cy0 DC Co0 Cx1 Cy1 DC Co1

8 sets of coordinate 2-D vectors (X, Y) are stored in R2 and R3 in the channel serial mode as

R2: X0 X1 … X7

R3: Y0 Y1 … Y7

The objective is to compute the following two line equations for each set of 2D coordinate and store the results in R4 and R5 as

R4: (X0*Cx0 + Y0*Cy0+Co0) … (X7*Cx0 + Y7*Cy0+Co0)

R5: (X0*Cx1 + Y0*Cy1+Co1) … (X7*Cx1 + Y7*Cy1+Co1)

Example LINE Equations

```
//-------------
// Example compute LINE equation in channel serial scenario
//-------------
line (8) acc:f  r1<0;1,0>:f  r2<0;8,1>:f// does acc = X# * Cx0 + Co0
```

mac (8) r4<1>:f r1.1<0;1,0>:f r3<0;8,1>:f// does r4.# = Y# * Cy0 + acc.#

line (8) acc:f r1<0;1,0>:f r2<0;8,1>:f// does acc = X# * Cx0 + Co0

mac (8) r4<1>:f r1.1<0;1,0>:f r3<0;8,1>:f// does r4.# = Y# * Cy0 + acc.#

The next example is to compute homogeneous dot product for OpenGL pixel shader running in Channel Serial. In this example, an original OpenGL PS instruction is like

dph R2.x R0 R1

With register remapping, we can store the input coefficient vector R0 in original format in r0, but 8 sets of input coordinate vectors in channel serial format in r2, r3, r4 and r5, and the destination R2.x component in r6.

r0: Cx0 Cy0 Cz0 Co0 DC DC DC DC

r2: X0 X1 … X7

r3: Y0 Y1 … Y7

r4: Z0 Z1 … Z7

r5: W0 W1 … W7

The objective is to compute the following DPH equations and store the results in r6 as

R6: (X0*Cx0+Y0*Cy0+Z0*Cz0+Co0) … (X7*Cx0+Y7*Cy0+Z7*Cz0+Co0)

Example Homogeneous Dot Product in Channel Serial

//-------------

// Example compute homogeneous dot product in channel serial scenario

//-------------

line (8) acc:f r0<0;1,0>:f r2<0;8,1>:f// does acc = X# * Cx0 + Co0

mac (8) acc:f r0.1<0;1,0>:f r3<0;8,1>:f// does acc.# = Y# * Cy0 + acc.#

mac (8) r6<1>:f r0.2<0;1,0>:f r4<0;8,1>:f// does r6.# = Z# * Cz0 + acc.#

Mask for SEND Instruction

Execution mask (upto 16 bits) for the SEND instruction is transferred to the Shared Function. This provides optimized implementation of DirectX Shader instructions.

Channel Enables for Extended Math Unit

The following example demonstrates how to use the SEND instruction to get service from the Extended Math unit.

Let's consider COS instruction in DirectX 10 in the following form

[([!]p0.{select|any|all})] cos[_sat] dest[.mask], [-]src0[_abs][.swizzle]

For a SIMD4x2 VS implementation with the following register mappings

p0 =>f0.0

src0 =>r0

dest =>r1

The equivalent GEN instruction is as the following

[([!]f0.0.{select|any4h|all4h})] SEND (8) r1[.mask]:f m0 [-][(abs)]r0[.swizzle]:f MATHBOX|COS[|SAT]

If the source swizzle is replication, the message description field can be modified to MATHBOX|COS|SCALAR to take advantage of the fast mode (scalar mode) supported by the Extended Math. The implied move of the SEND instruction is equivalent to the following instruction:

MOV (8) m0[.mask]:f [-][(abs)]r0.0[.swizzle]:f {NoMask}

For a SIMD16 PS implementation, the register mappings are as the followings

p0 =>f0…f3 // in order of R, G, B, A

src0 =>r0,r1; r2,r3; r4,r5; r6,r7

dest =>r8,r9; r10,r11; r12,r13; r14,r15

There are several ways to translate the DirectX instruction, depending on the operand/instruction modifiers present in the DirectX instruction. If predicate is not present and the source swizzle is replication, say, src0.y, which is r2-r3, the translation could be as the following instructions

send (8) r8:f m0 -(abs)r2:f MATHBOX|COS

send (8) r9:f m1 -(abs)r3:f MATHBOX|COS {SecHalf}// use the second half of 8 flag bits

mov (16) r10:fr8:f // All destination color chan's are same

mov (16) r12:fr8:f // MOV is faster than most MathBox func's

mov (16) r14:fr8:f // These MOV's are compressed instructions

Notice that instead of issuing Extended Math messages with the same input data, destination color channel replication is performed by the MOV instructions. This is faster for the thread for most cases as many Extended Math functions consume multiple cycles. This also conserves message bus bandwidth as well as the usage of the shared resource – Extended Math. The destination mask in the DirectX 10 instruction indicates which of the r8 to r15 registers are updated. If the source swizzle is not replication, there will be 8 SEND instructions.

With predication on, if the predication modifier is p0.select, translation is to take the selected flag register f#. The other predication modifiers '.any' and '.all' are translated into '.any4v' and '.all4v', respectively. Notice that with predication on, it is not required to run all 4 pixels in a subspan in the same way, so no need to enforce .any4h/.any4v. The following example shows the instruction with predication (but without .select modifier).

(f0[.any4v|.all4v]) send (8) r8:f m0 -(abs)r2:f MATHBOX|COS

(f0[.any4v|.all4v]) send (8) r9:f m1 -(abs)r3:f MATHBOX|COS {SecHalf}

(f1[.any4v|.all4v]) mov (16) r10:fr8:f // All destination color chan's are same

(f2[.any4v|.all4v]) mov (16) r12:fr8:f // MOV is faster than most MathBox func's

(f3[.any4v|.all4v]) mov (16) r14:fr8:f // These MOV's are compressed instructions

The same instructions works also for predication with select component modifier. We simply replase f0 to f3 above by the selected flag register, say, f1. The modifier of any4h/all4v would also work.

Channel Enables for Scratch Memory

The following example demonstrates how to use the SEND instruction to get service from the Data Port for scratch memory access.

Let's consider general instruction in DirectX 10 that uses scratch memory as a source operand

[([!]p0.{select|any|all})] add dest[.mask], [-]src0[_abs][.swizzle], [-]src1[_abs][.swizzle]

For a SIMD4x2 VS implementation with the following register mappings

p0 =>f0

src0 =>r0

src1 =>s2 / r10

dest =>r1

In this example, the scratch memory offset is provided by an immediate and a GRF register r10 is used as the intermediate GRF location for spill/fill of scratch buffer accesses. This arithmetic instruction is converted into a Data Port read followed by an arithmetic instruction.

mov (8) r3:d r0:d {NoMask}// move scratch base address to be assembled with offset values

mov (1) r3.0:d 2*32 {NoMask}// s2 for vertex 0

mov (1) r3.1:d 2*32+16 {NoMask}// s2 for vertex 1

send (8) r10 m0 r3 DATAPORT|RC|READ_SIMD2

[([!]f0.{sel|any4h|all4h})] add (8) r1[.mask]:f [-][(abs)]r0[.swizzle]:f [-][(abs)]r10[.swizzle]:f

So if scratch register is the source, there is no need to use the channel enable side band. This is also true for channel-serial PS cases.

Now, let's consider the case when a scratch register is the destination of an instruction.

p0 =>f0

src0 =>r0

src1 =>r1

dest =>s2 / r10

We have

add (8) m1:f [-][(abs)]r0[.swizzle]:f [-][(abs)]r1[.swizzle]:f

mov (8) r3:d r0:d {NoMask}// move scratch base address to be assembled with offset values

mov (1) r3.0:d 2*32 {NoMask}// s2 for vertex 0

mov (1) r3.1:d 2*32+16 {NoMask}// s2 for vertex 1

```
[([!]f0.{sel|any4h|all4h})] send (8) null[.mask] m0 r3 DATAPORT|RC|WRITE_SIMD2
```

Notice that with a null as the posted destination register, we are able to transfer the [.mask] over the message channel enables. In many cases for scratch memory assess, a write-with-commit is required, therefore, the posted destination register could be r10.

Now, let's consider the PS case when a scratch register is the destination of an instruction.

p0 =>f0-f4

src0 =>r0-r7

src1 =>r8-r15

dest =>s16-s23 / r16-r23

When predication is not on (or predication with swizzle control on), we have

add (16) m4:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m6:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m8:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m10:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (8) r3:d 0x76543210:v {NoMask}// ramp function

mul (16) acc0:d r3:d 16 {NoMask}// ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf}// ramp function

add (16) m2:d acc0:d 2*256 {NoMask}// ramp function

```
send (16) null m1 r3 DATAPORT|RC|WRITE_SIMD16
```

As there is no bit left from the unit specified descriptor field, the 4 bit mask must be put into the header field in m1, which requires at least two more instructions.

Alternatively, or for the case that predication without modifier is on, we can do a read-modify-write.

mov (8) r3:d 0x76543210:v {NoMask}// ramp function

mul (16) acc0:d r3:d 16 {NoMask}// ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf}// ramp function

add (16) m2:d acc0:d 2*256 {NoMask}// ramp function

```
send (16) r16 m1 r3 DATAPORT|RC|READ_SIMD16 // read from scratch
```

// some of the following four instructions may be omitted based on [.mask] field

[([!]f0.{sel|any4v|all4v})] add (16) r16:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r18:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r20:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r22:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (16) m4:f r16:f {NoMask}

mov (16) m6:f r18:f {NoMask}

mov (16) m8:f r20:f {NoMask}

mov (16) m10:f r22:f {NoMask}

```
send (16) null m1 null DATAPORT|RC|WRITE_SIMD16 {NoMask}// write back to scratch
```

Flow Control Instructions

Unconditional branches are performed through direct manipulation of the 32-bit IP architectural register. For example:

```
mov (1) IP <memory_address>// jump absoluteadd (1) IP IP <byte_count>// jump relative
```

Note that jump distances are specified in terms of bytes, as opposed to instruction counts in the case of *break*, *halt*, etc. To minimize confusion, an assembler-only instruction 'jmp <inst_count>', where <inst_count> is an immediate term, may be defined which takes an instruction count for a distance. The *jmp* pseudo-opcode can be mapped to an "add (1) ip ip <inst_count> * 16" instruction.

IP is aligned to an 8-byte boundary, thus the 3 LSBs are not maintained in the IP architectural register and should not be relied upon by software.

IP, when used as a source operand, reflects the memory address of the instruction in which it is used. The following are examples illustrating the use of IP:

```
add (1) IP4*16// jumps to HERE_1
add (1) IP0x35// jumps to HERE_1 (4 lsb's don't-care)  <instruction>
 <instruction>

HERE_1:<instruction>HERE_2:<instruction>
 <instruction>
 add (1) IP -2*16// jumps to HERE_2 ...
 add (1) IP 0// infinite loopadd (1) IP 0xF// infinite loop ...
```

Note for Assembler: The if/iff/else/while/break instructions identify relative addresses as the targets of an implicit jump associated with the instruction. These are optional in the assembly syntax as the jitter can determine the location of the matching instruction (e.g. matching endif instruction for a given if instruction).

Execution Masking

This topic is currently under development.

Branching

Example. If / Else / EndIf

```
//-------------
// Example if/else/endif scenario
// "if (r5==r4) ...else ... end-if"
//-------------
...
cmp.e.f0 (8) null r5 r4// does r5 == r4?
(f0) if (8)  HERE_1// "if" part - save then update IMASK;
//   or goto the 'else' if all false
...
...
HERE_1:// now do the 'else' part
else (8) HERE_2// "else" part - invert IMASK
//   or goto the 'endif' if all false
...
...
HERE_2:
endif// "end-if" part – restore IMASK
....// and continue...
```

If it is known that the code has no nested conditionals, a predicate can be used for a lower overhead, more efficient if/else/endif. (One must consider the probability of all channels taking the same branch, and the number of instructions under the if/else blocks as to which conditional method, predicate or mask, is most efficient).

Fast-If

Below is an example of a fast-if instruction. For the 'iff' instruction, only and iff-endif construct is allowed, as opposed to a if-else-endif. Note that the target address for branching if all enabled channels fail is one instruction beyond the endif, as the 'iff' does not push and update the IMask unless the branch is taken for at least one execution channel.

Example Fast If

```
//-------------
// Example – Fast If
//One instruction overhead conditional
//-------------
...
cmp.e.f0 (8) null r5 r4// any flag update
...
(f0)iff (8) HERE_1// "fast-if" – only pushes IMask;
//   if execution falls through,
//   else go to HERE_1
...
...
endif// "end-if" part – restores IMask
HERE_1:
...// and continue...
```

Cascade Branching

As there is no 'elseif' instruction, a C-like cascade branching such as if / elseif / else / endif, can be realized using the basic building blocks of if / else / endif as shown in the following example. Notice that two 'endif's' are required in order to pop the IStack correctly.

Example. If / Elseif / Else / EndIf

```
//-------------
// Example if/elseif/else/endif scenario
// "if (r5==r4) ...elseif (r6>r7) else ... end-if"
//-------------
...
cmp.e.f0 (8) null r5 r4// does r5 == r4?
(f0)if (8) HERE_1// "if" part - save then update IMask;
//   or go to the 'else' part if all false
...
...
```

```
HERE_1:// now do the ‘else’ part
else (8) HERE_2// “else if” part - invert IMask
//   or go to the ‘else’ part if all false
cmp.g.f0 (8) null r6 r7// is r6 > r7?
(f0)if (8) HERE_3// “if” part - save then update IMask;
//   or go to the ‘else’ part if all false
...
...
HERE_3:// now do the ‘else’ part
else (8) HERE_4// “else” part - invert IMask
//   or go to the ‘end-if’ part if all false
...
...
HERE_4:
endif// “end-if” part – restore IMask for elseif
HERE_2:
endif// “end-if” part – restore IMask for if
....
```

Compound Branches

Compound branches are supported through the ability logically combine flag registers for each intermediate result.

Example Compound Branch

```
//-------------
// Example: “if (r0 > r1) OR (r2 <= r3)”
//-------------
...
cmp.g.f0 (8) null r0:d r1:d// r0 > r1?
cmp.le.f1 (8) null r2:d r3:d// r2 <= r3?
or (1) f0:w f0:w f1:w// combine f0 and f1
(f0) if (8) HERE_1// Can now do normal if/else
```

...

...

HERE_1:endif

...

Example Compound Branch Using 'Any' or 'All'

```
//-------------
// Example: assuming we're doing a channel-serial vector in r0-r3
// We want to know if all components of the vector are > 0x80
//-------------
...
cmp.g.f0 (16)  null r0 0x80// r0 > 0x80?
cmp.g.f1 (16)  null r1 0x80// r1 > 0x80?
cmp.g.f2 (16)  null r2 0x80// r0 > 0x80?
cmp.g.f3 (16)  null r3 0x80// r1 > 0x80?
(f0.all4v) if (16) HERE_1
...
...// code executed only for those channels
...// where per-channel r0,r1,r2,r3 all > 0x80
...
HERE_1:endif
...// and continue...
```

Looping

Due to GEN's SIMD-16 architecture, it must support the case of up to 16 loops running in parallel. These must be handled as independent loops, each with its own loop-exit condition which could occur after a different number of loop iterations. To account for each channel's progress, a 16b loop-mask 'LMask' is defined with 1b associated to each execution channel. This mask keeps track of which channels remain active inside a loop block.

Basic Do-While Loop

Looping illustrates the most basic loop. Two operations must be accomplished before loop entry. (1) Prior to loop entry, there is some subset of enabled channels as dictated by the code sequence prior. In general, the active status of each channel is indicated in the virtual EMask any point in time. These active channels will become the channels over which the loop is run, and LMask must be initialized with the

EMask value. (2) Since a given loop may be nested within another loop, the previous LMask & CMask must be saved to the LStack for later restoration upon loop completion. The 'msave' instruction performs both the save and update in a single instruction, and thus all loop-blocks should be fronted with a "msave LStack LMask" and "msave LStack CMask" operation.

Note that the LMask and CMask share the same mask-stack. Thus, CMask must always be a 1's-subset of the LMask for proper stack operation. This is the case if CMask is updated to LMask each pass through the loop (see *Looping*) and through the 'break' instruction updating both masks.

Each pass through the loop, a loop terminating operation must be evaluated and stored in a flag register. This condition must be evaluated on a channel-by-channel basis as exemplified:

cmp.z.f0(8) null r2 d3// any operation that updates a flag

The result of this operation sets a bit per channel in the specified flag register, which is then used in the 'while' instruction. As loops are performed, channels may become disabled as their termination condition is met.

'While' termination is determined on a channel-by-channel basis by the logical AND of corresponding bit positions of AMask, CMask and the specified flag. If the result is '1' the channel remains enabled for the next pass of the loop; if '0' the channel is disabled until loop fall-through. The 'while' instruction causes the LMask to be updated with the latest result of enabled channels. If any channel remains enabled (LMask != ...000b), an additional pass through the loop is made. Once a channel is terminated for the loop operation, it remains terminated until the loop is complete for all channels.

Upon fall through, the 'while' instruction causes the previously saved LMask & CMask to be popped from the LStack, enabling execution on the same subset of channels enabled prior to loop entry (unless a channel had been otherwise terminate inside the loop via 'halt').

Example Basic Loop Construct

```
//-----------------------
//Example: Basic do-while loop structure
//-----------------------
...
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask LMask{NoMask}// update CMask for this pass
...
...
<some flag update>
()while (8) BEGIN_LOOP// cond. branch
//   + restores LMask on fall-through
```

...

Do-While Loop with Break

A loop may also be terminated for any channel via the 'break' instruction. The 'break' instruction causes the corresponding bit positions of enabled channels to be cleared in the LMask. If the updated LMask = ...000b, a branch is made to the specified instruction location. An example is shown below in which the 'break' is at the same conditional-nesting level as the terminating 'while'. Its primary value may simply be to support a "do...break.. while (true)" –type structure for a more direct 1:1 translation from higher-level source code.

Example Loop Construct With Non-Nested 'Break'

```
//-------
//Example: While-true loop
//-------
#define BrkCode(i,d)(i « 16) + d
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask LMask{NoMask} // update CMask for this pass
...
<some flag update>
()break (8) BrkCode(0,HERE_1)// Restores LMask when all
// channels complete loop.
...
...
while (8) BEGIN_LOOP// while true
HERE_1:
...
```

A break condition may occur from various levels of nested-ifs. This gives rise to the possibility that a the loop may terminate from within nested 'if's, and due to the jump inherent in the 'break' instruction, the associated 'endif's are not encountered to clean-up the IStack as nesting levels are exited.

Example Loop Construct With 'Break' From Within Nested If's

```
//-------
//Example: General Loop Structure w/ break inside if's
//-------
```

```
#define BrkCode(i,d)(i « 16) + d
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask LMask{NoMask} // update CMask for this pass
...
if ...
if ...
if ...
...
()break (8) BrkCode(3,HERE_1)// we're 3 levels deep, so
...
endif
endif
endif
...
()break (8) BrkCode(0,HERE_1)
...
while (8) <flag_spec> BEGIN_LOOP// cond. branch
// + restores C/LMask on fall-through
HERE_1:
```

Do-While Loop with Continue

A continue instruction 'cont' is provided skip to the next iteration of the loop. Because not all channels participating in the loop may be enabled at the time this instruction is executed, some channels may require continuation of the loop. A special mask 'CMask' is defined which accounts for channels temporarily disabled for the current loop pass.

Since loops may nested, the CMask must be saved and restored around a loop similar to LMask. Since the CMask value within a properly constructed loop is always a subset of the LMask, it can share the LStack for storage, so long as it is pushed after LMask as shown in *Looping*. This save/restore operations are not required if the loop being entered does not have any occurrence of a continue instruction.

Example Do-While with Continue

```
//-------
//Example: General Loop Structure w/ basic break and cont.
```

```
//-------
#define ContCode(i,d)(i « 16) + d
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask EMask// re-initialize CMask for this pass
...
...
() cont (8) ContCode(0,HERE_1)
...
HERE_1:
()while (8) BEGIN_LOOP// cond. branch
//   + restores C/LMask on fall-through
...
```

Indexed Jump

Example Indexed Jump

```
//-------------
// Code example shows the use of jmpi to perform a case statement
// of any number of options in 3 jumps
//-------------
.default_execution_size 8
...
jmpi r0<0,1,0>// jump relative, based on r0.a.x
// ----- Jump Table ------
jmp HERE_0// redirect for case 0
jmp HERE_1// redirect for case 1
jmp HERE_2// redirect for case 2
jmp HERE_3// redirect for case 3
...
HERE_0:// ... case 0 ...
```

...

jmp DONE

HERE_1:// ... case 1 ...

...

jmp DONE

HERE_2:// ... case 2 ...

...

jmp DONE

HERE_3:// ... case 3 ...

...

DONE:

...// and continue...

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 8: Media VDBOX

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

VDBOX Registers **1**

MFX Architecture **1**

MFX Introduction 1

MFC Overview 2

Sample Algorithmic Flow 3

Synchronization Mechanism 4

Restrictions 5

MFD Overview 6

MFD Memory Interface 10

MFD Codec-Specific Commands 11

MFX State Model 11

MFX Interruptability Model 12

Decoder Input Bitstream Formats 13

AVC Bitstream Formats – DXVA Short 13

AVC Bitstream Formats – DXVA Long 13

VC1 Bitstream Formats – Intel Long 13

MPEG2 Bitstream Formats – DXVA1 13

JPEG Bitstream Formats – Intel 13

Concurrent Multiple Video Stream Decoding Support 14

MFX Codec Commands Summary **14**

MFX Decoder Commands Sequence 18

Examples for AVC 18

Examples for VC1 21

Examples for JPEG 22

MFX Pipe Common Commands **23**

Bitplane Buffer 25

Video Codecs **26**

AVC (H.264) 27

AVC Common Commands 27

AVC Decoder Commands 27

Session Decoder StreamOut Data Structure 28

AVC Encoder PAK Commands 37
PAK Object Inline Data Description 38
Luma Intra Prediction Modes 51
Reference Indices Defined for Each MB Partition Type and Bit Assignment 55
MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags) 55
Macroblock Type for Intra Cases 60
Macroblock Type for Inter Cases 64
Macroblock Type Conversion Rules 68
Indirect Data Description 73
Unpacked Motion Vector Data Block 74
Packed-Size Motion Vector Data Block 78
Macroblock Level Rate Control 80
Theory of Operation Overview 82
AVC Encoder MBAFF Support 83
MPEG-2 84
MPEG2 Common Commands 84
MPEG2 Decoder Commands 85
Indirect Data Description 85
MPEG2 Encoder PAK Commands 86
PAK Object Inline Data Description – MPEG-2 86
MFX HW Interface and DXVA Conversion 93
Map DXVA to HW BSpec 93
Map HW Bspec to DXVA 95
Video Codec VC-1 97
VC1 Common Commands 97
VC1 Decoder Commands 97
Handling Emulation Bytes 98
VP8 99
MFX_VP8_STATISTICS - Encoder Only 99
VP8 Common Commands 101
VP8 Decoder Commands 102
VP8 Encoder Commands 102
JPEG and MJPEG 103

JPEG Decoder Commands 103
JPEG Encoder Commands 106
More Decoder and Encoder 109
MFD IT Mode Decode Commands 109
Common Indirect IT-COEFF Data Structure 109
Inline Data Description in MPEG2-IT Mode 110
Indirect Data Format in MPEG2-IT Mode 113
MFX Deblocking Commands 113
Encoder StreamOut Mode Data Structure Definition 114
PAK Frame Statistics StreamOut 115
PAK Multi-Pass 116
Driver Usage 117
Programming Reference 117
Monochrome Picture Processing 117
Context Switch 118
PMSI Support 118
Pipeline Flush 118
MMIO Interface 118
Decoder Registers 119
Encoder Registers 120
MMIO Interface 121
Row Store Sizes and Allocations 123

VDBOX Registers

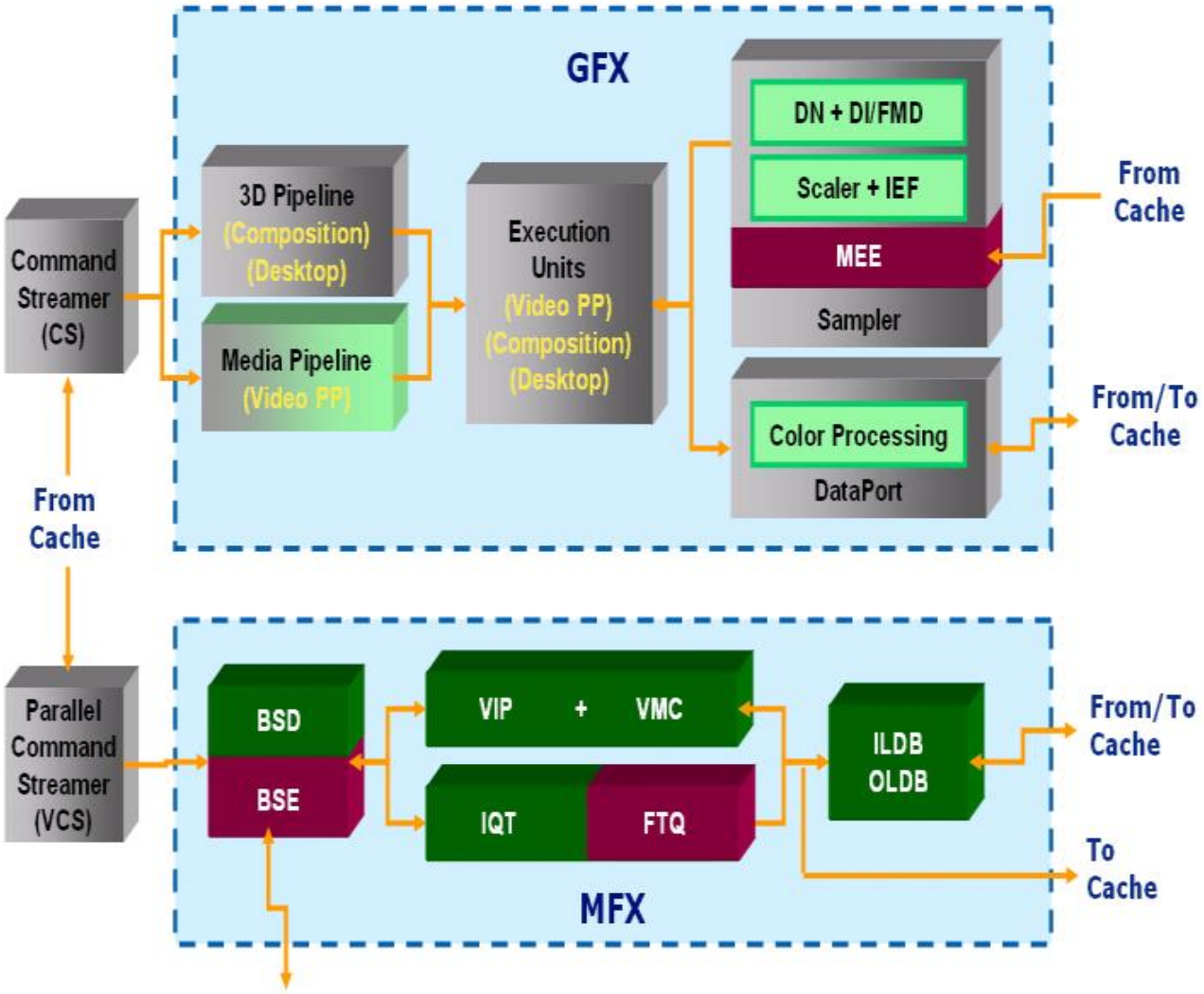
This section describes the VDBOX Command Memory Interface registers.

MFX Architecture

This section and the following sections of Media VDBOX contain the referential documentation on the Multi-Format Codecs, or MFX for those series of chips.

MFX Introduction

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).



MFC Overview

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).

Note: For Gen6, MFC only supports AVC (H.264).

Many decoding function blocks in MFD such as VIP, VMC, IQT, etc, are also used in encoding mode. Two blocks, FTQ and BSE, are encoding only.

The encoding process is partitioned across host software, the GPE engine, and the MFX engine. The generation of transport layer, sequence layer, picture layer, and slice header layer must be done in the host software. GP hardware is responsible for compressing from Slice Data Layer down to all macro-block and block layers. Specifically, GPE w/ VME acceleration is for motion vector estimation, motion estimation, and code decision.

MFC is operated concurrently with and independently from the GPE (3D/Media) pipeline with a separate command streamer. The two parallel engines have similar command protocol. They can be executed in parallel with different context. For encoding, motion search, MB mode decision, and rate control are performed using GPE pipeline resources.

MFC is implemented to achieve the following objectives:

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
    - Support AVC 4:2:0 Main Profile and High Profile only (8-bit only), up to Level 4.1 resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be encoded. There is no support for Baseline, Extended, or High-10 Profiles.
- Performance requirements with MFX core frequency above 667MHz:
    - Real-time performance with 20% duty cycle or less.
    - Support concurrent decoding of two active HD bitstreams of different formats (for example, one AVC and one VC1 HD bitstream) and one active HD encoding.

As the result of this hardware partitioning, VPP and ENC are always running in GPE, and PAK is what runs exactly in MFC.

PAK – residue packing and entropy coding, including block transformation, quantization, data prediction, bitrate tuning and reference decoding. It delivers final packed bitstream and decoded key-frame reference:

- As the same as ENC, PAK is invoked on a Slice boundary; a single call of VPP can lead to multiple calls for PAK.
- Rate control is inside ENC and PAK only, not in VPP.
- PAK must always perform with reconstructed reference picture.

There is a general dependency of the three operation pipelines. Semaphores are inserted either according to frames or slices. The main CS will also be notified when the decoded reference is ready for the next frame set to be encoded. The detailed discussion will be found in a later section.

Host software is responsible for encoding the transport stream and all the sequence, picture, and slice layer/header in the bit-stream; the MFC system is responsible for compressing from Slice Data Layer down to all macro-block and block layers.

Sample Algorithmic Flow

Assuming all the hardware components are given, there are infinite usage possibilities left with intention for software to decide according to its own application needs depending upon the balanced requirement of coding speed, frame latency, power-consumption, and video quality, and depending upon the usage modes and user preferences (such as low-frame-rate-high-frame-quality vs. high-frame-rate-low-frame-quality).

The last part of this chapter, we illustrate a generic sample to show how a compression algorithm can be implemented to use our hardware.

Step 1. Application or driver initializes the encoder with desired configuration, including speed, quality, targeted bit-rate, input video info, and output format and restrictions.

Step 2. VPP – Application or driver feeds VPP one frame at a time in coded order with specified frame or field type, as well as transcoding information: motion vectors, coded complexity (i.e. bit size).

It will perform denoising and deblocking based on original and targeted bit-rate, and output additional

4 spatial variances and 2 temporal variances for each macroblock as well as the whole frame.

Step 3. ENC – Application or driver feeds ENC one coding slice buffer at a time including all VPP output. The frame level data is accessible to all slices.

a. Encoding setup unit (**ESE**) will set picture level quality parameters (including LUTs, and other costing functions) and set target bit-budget (TBB) and maximal bit-budget (MBB) to each macroblock based on rate-control (**RC**) scheme implemented. For B-frames, it will also make ME searching mode decision (either Fast, Slow or Uni-directional).
b. Loop over all macroblocks: calculate searching center (**MVP**) perform individual ME and IE (**MEE**). Multi-thread may be designed for HW according to a zigzag order for minimal dependency issue.
c. ENC make microblock level code decision (**CD**) outputs macroblock type, intra-mode, motion-vectors, distortions, as well as TBBs and MBBs.

Step 4. PAK – Application or driver feeds PAK one array of coded macroblocks covering a slice at a time, including all ENC output. Original frame buffer and reconstructed reference frame buffers are also available for PAK to access.

a. PAK may create bitstreams for all sequence, gop, picture, and slice level headers prior the first macroblock.
b. Loop over all macroblocks, accurate prediction block is constructed for either inter- or intra-predictions (**VMC** & **VIP**). If MB distortion is less than some predetermined threshold, for a B slice

this step can be skipped as well as the Steps (c)-(e) and jump directly to Step (f); for a key slice the prediction calculated here will be directly used as the reference thus it jumps to Step (e) after this step.

c. Differencing the predicted block from the original block derives the residue block. Forward transformation and quantization (**FTQ**) is performed. For B slice, it will jump to Step (f) right after. For other types of slice, Steps (d) and (e) can be performed in a thread in parallel with Step (f) and beyond.
d. This is for accurate construction of reference pictures. Inverse quantization and inverse transformation (**IQT**) are performed and added to the predictions to have the decoded blocks.
e. **ILDB** is applied accordingly to the reconstructed blocks.
f. Meanwhile macroblock codes: including its configuration info (types and modes), motion info (motion vectors and reference ids), and residual info (quantized coefficients), are collected for packing (**BSE**) in the following sub-steps:
    i. Code clean-up (in **MPR**). Check and verify Mbtype and Cbps, use Skip or Zero respectively if one can. In principal, when there are equivalent codes, use the simple one.
    ii. Drop dependency (in **MPR**). Calculate relative codes from the absolute codes by associate them with neighborhood information. All neighborhood correlations are solved in this step.
    iii. Unify symbols (in **SEC**). Translate relative codes into symbols, and table or context indices that are independent of the concept of syntax type.
    iv. Entropy coding (**VLE**) on symbols.
g. Parsing bitstream data in RBSP form (in **VLE**), and output to application or driver.
h. By the end of each picture, write out the accurate actual data size to designate buffer for ENC to access.

Synchronization Mechanism

Encoding of a video stream can be broken down to three major steps (as explained in the previous section):

1. VPP: video-stream pre-processing
2. ENC: encoding, *that is,* code decision of inter-MVs and intra-modes
3. PAK: bit-stream packing
    a. residual calculation, transformation, and quantization
    b. code bit-stream packing
    c. reference generation of keyframes

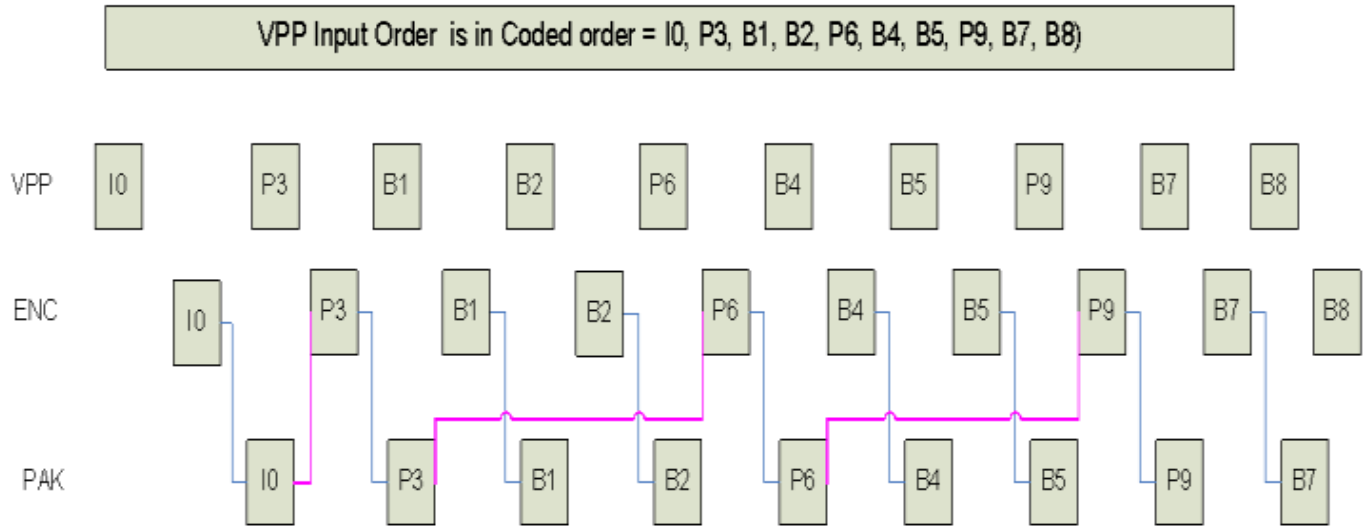
This section describes an architectural solution to map the first two steps in the GFX engine and the last step in the MFX engine. Since this mapping involves two OS-visible engines, managing them in parallel under one application is similar to the solution in earlier generations. Each engine has its own command streamers and has mechanisms to synchronize at required levels as described in the next sub-section.

Above three steps of encoding have dependencies in processing based on

i. Functional pipeline order, *i.e.* on a given frame, VPP needs to be performed first, then ENC, then PAK and finally MFD (*Multi-Format Decoding*) for key reference frame generation.
ii. I-frames are key frames for P and B, they have to be first in every pipe-stage.
iii. P-frames are key frames for B frames and therefore P frames are processed first before the dependent B frames
iv. GFX Engine is time slice to work on either VPP or ENC frame as we discussed in the previous chapter.
v. PAK + MFD are executed on the same frame in the MFX engine by macro-block level pipelining within a slice. It should be noted that for the sake of simplicity, an entire frame (potentially multiple slices) are processed in the corresponding engine and no smaller granularity of switching is allowed between the functional pipeline stages.

Three steps of the encoding can be interleaved on two engines in the following way on a frame by frame basis.

Command Stream Synchronization

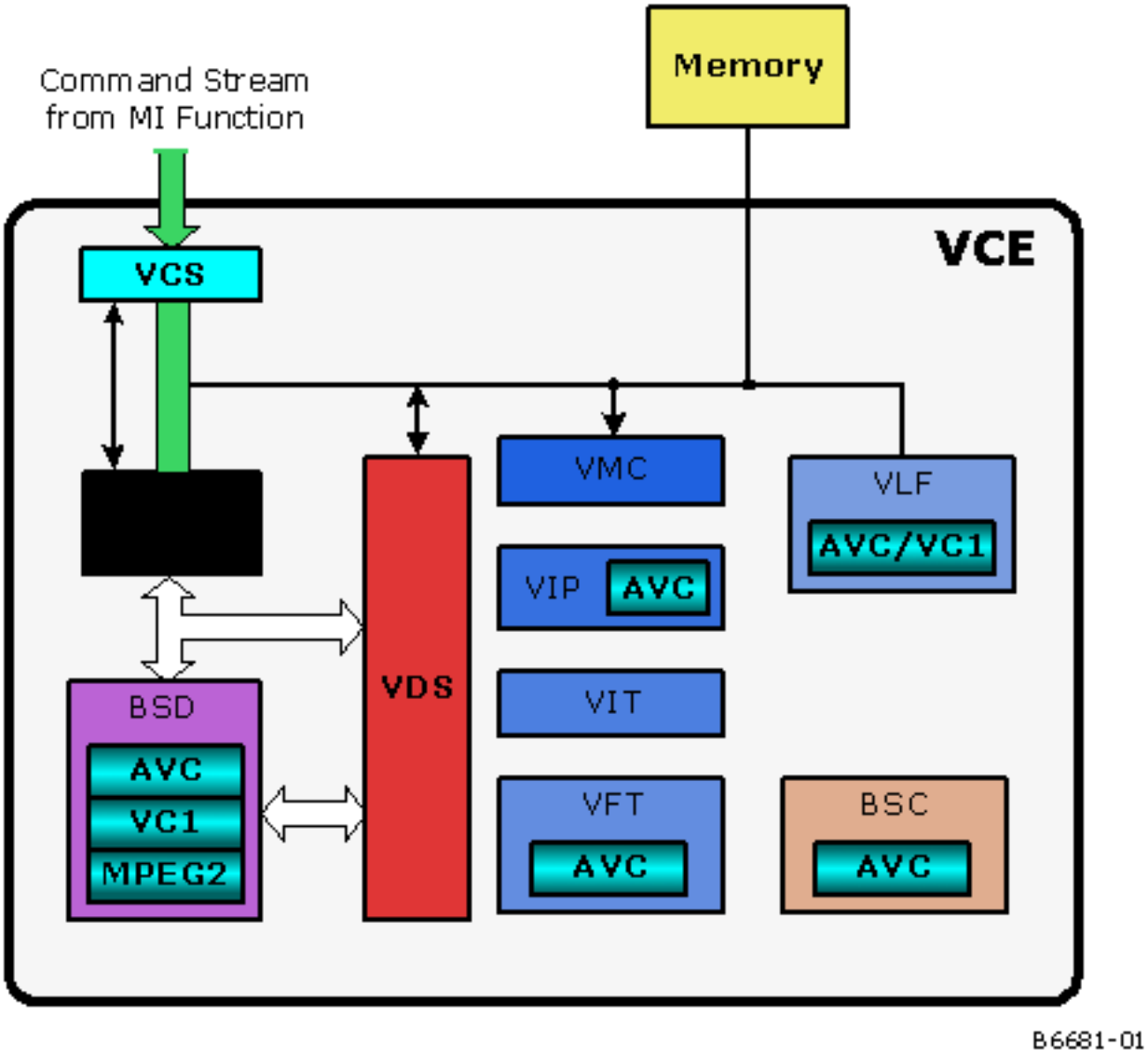


Restrictions

MFC implementation is subject to the following limitations.

- Context switching within MFC and with Graphics Engine occurs only at frame boundary to minimize the amount of information that needs to be tracked and maintained.

MFD Overview



When used for decoding, we also refer to the MFX Engine as the MFD Engine.

The Multi-Format Decoder (MFD) is a hardware fixed function pipeline for decoding the three video codec format and one image compression codec format: AVC (H.264), VC-1, MPEG2, and JPEG.

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
  - Support AVC 4:2:0 Main and High (8-bit only) Profile only (no support for Baseline, Extended and High-10 Profiles), up to Level 5.1 (max 983,040 MB/s, max 36,864 MB/frame, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
    - Allow a B-picture (frame or field) as a reference picture

- Support MVC 4:2:0 Stereoscopic Progressive Profile only, up to Level 5.1 (max 983,040 MB/s per view, max 36,864 MB/frame per view, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
  - Does not support interlaced video specified in the Stereoscopic Profile
- Support VC1 4:2:0 Simple, Main and Advanced Profiles, up to Level 4 (max 491,520 MB/s and max 16,384 MB/frame; max only one dimension will be at 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
  - Allow a B-field as a reference picture only in interlaced field decoding, no other modes.
- Support MPEG2 HD Main Profile (4:2:0), up to High Level (1920x1152 pixels) and up to 80 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded. No support for SNR and spatial-scalability.
  - Does not support B-picture as a reference picture.
- Support Baseline JPEG with five choma types (4:0:0, 4:1:1, 4:2:2, 4:2:0, and 4:4:4. No support for Extended DCT-based mode, Progressive mode, Loseless mode, nor Hierarchical mode.
  - H/W support 64Kx64K, but Surface State can support only up to 16kx16k

Features	Supported	Unsupported
Coding processes	Baseline sequential mode: • 8-bit pixel precision of source images • loadable 2 AC and 2 DC Huffman tables • 3 loadable quantization matrix for Y, U, V • Interleaved and non-interleaved Scans • Single and multiple Scans	Extended DCT-based mode, Lossless, Hierarchical modes: More than 8 bit pixel resolution, progressive mode, arithmetic coding, 4 AC and 4 DC Huffman tables (extended mode), predictive process (lossless), multiple frames (hierarchical)
Number of image channels	1 for grey image 3 for Y, Cb, Cr color image	4-th channel (usually alpha blending image)
Image resolution	Arbitrary image size up to 16K * 16K	Larger than 16K * 16K (64K * 64K is max. in the JPEG standard)

Features	Supported	Unsupported
Chroma subsampling ratio	Chroma 4:0:0 (grey image) Chroma 4:1:1 Chroma 4:2:0 Chroma horizontal 4:2:2 Chroma vertical 4:2:2 Chroma 4:4:4	Any other arbitrary ratio, e.g., 3:1 subsampled chroma
Additional feature (post-processing)	Image rotation: 90/180/270 degrees	

    - H/W does not impose restriction on picture frame aspect ratio, but is bounded by a max 256 MBs (4096 pixels) per dimension programmable at the H/W interface specifications.
        - For example, supporting HD video resolution 1920x1080/60i, 1920x1080/24p, 1280x720/60p
- Performance requirements with MFX core frequency above 1GHz
    - Real-time performance around 10% duty cycle
    - Support concurrently decoding of at least two active HD bitstreams of different formats (For example, one AVC and one VC1 HD bitstream)
- The parsing of transport layer and sequence layer is not performed in this hardware, and is required to be done in the host software.
- The MFD hardware pipeline is operated concurrently with and independently from the Graphics (3D/Media) pipeline with separate command streamer. The two parallel engines are designed with the similar command protocol. They can be executed in parallel with different context.
- Local storages and buffers along the hardware pipeline are kept at minimum. For example, there is no on-die row-store memory. They are resided on the system memory. MFD is designed to hide the memory access latency (in both the row stores and in the motion compensation units) in maximizing its decoding throughput.
- Support the following operating modes:
    - VLD mode - operation starts from entropy decoding of the compressed bit stream (parsing Slice Header and Slice Data Layer in AVC, Picture layer, Slice layer and MB Layer in VC-1, and MB-layer in MPEG2), all the way, to the reconstruction of display picture, including in-loop and out-loop deblocking, if any.
        - Streamout mode - a new feature of the VLD mode in assisting transcoding during decoding. Selected uncompressed data (e.g. per MB MV information) will be sent out to the EU and the ME engine (resided on the Sampler of the 3D Gx Pipeline) for encoding into a different format or for the purpose of transcaling and transrating. In addition, the uncompressed result may continue to be processed by the rest of pipeline as in VLD mode to generate the display picture for transcoding. That is, while

intermediate data are streaming out to the memory, the MFD Engine continues its decoding as ususal.

- For JPEG, only VLD mode is supported (No IT mode). Host software decodes Frame and Scan layers (down to Scan header in the JPEG bit stream syntax) and sends all the corresponding information and Scan payload to the MFD hardware pipeline.

  - IT mode - when host software has already performed all the bit stream parsing of the compressed data and packaging the uncompressed result into a specific format (as a sequence of per-MB record) stored in memory. The hardware pipeline will fetch one MB record at a time and perform the rest of the decoding process as in VLD mode
  - Host software (Application) is responsible for parsing and decoding all the transport and program layers, and all sequence layers. These parameters are passed to Driver and forwarded to H/W as needed through different STATE commands. Host software is also responsible for separating non-video data (audio, meta and user data) from sending to H/W.
    - MFD Engine is only responsible for macro-block and block layers decoding, plus certain level of header decoding. For AVC MFD starts decoding from Slice Header; for VC1, MFD starts decoding from Picture Header, and for MPEG2 decoding starts from MB Layer only.
    - For JPEG, MFD is responsible for ECS and block layers decoding.

- Support bitstream formats (compressed video data) for each codec
  - AVC - 2 formats
  - MVC - 2 formats
    - DXVA2 MVC Short Slice Format
    - DXVA2 AVC Long Slice Format Specification (exactly the same as AVC)
  - VC1 - 2 formats
    - Fully compliant to Picture Parameter and Slice Control Parameter interface definition
  - MPEG2
    - MB Layer only, according to DXVA 1 Specification
  - JPEG
    - ECS Layer
- The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.
  - In particularly, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.
- To activate the AVC deblocker logic for incoming uncompressed 4:2:0-only video stream, one can pack the uncompressed video stream to compliant with the IPCM MB data format (including ILDB

control information) and feed them into the MFD engine in IT mode. Since the MFD Engine is in IPCM mode, transformation, inter and intra processing are all inactive.

Start Code Detection and removal are done in the CPU, but the Start Code Emulation Prevention Byte is detected and removed by the front end logic in the MFD. The bitstream format for each codec and for each mode is specified in this document.

Codec specific information are based on the following released documents from third parties:

- Draft of Version 4 of H.264/AVC (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding); JVT-O205d1.doc; dated 2005-05-30
- Final Draft SMPTE Standard : VC1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M, dated 2006-1-6; PDF file.
- MPEG2 Recommendation ITU T H.262 (1995 E), ISO/IEC 13818-2: 1995 (E); doc file.
- Digital Compression and Coding of Continuous-tone Still Images, ITU-T Rec. T.81 and ISO/IEC 10918-1: Requirements and guidelines September 18 1992; itu-t81[1].pdf

MFD Memory Interface

The Memory Arbitrator follows the pre-defined arbitration policy (as indicated in the following listing P0 to P11, in which P0 is the highest priority) to select the next memory request to service, then it will perform the TLB translation (translation to physical address in memory), and make the actual request to memory.

The Memory Arbitration unit is also responsible for capturing the return data from memory (read request) and forward it to the appropriate unit along the MFD Engine.

- Read streams: (all 64B requests)
    - Commands for BSD : linear (including indirect data) (P0)
    - Indirect DMA (P1)
    - Row store for BSD: linear (P5)
    - Row store for MPR: linear (P6)
    - MC ref cache fetch : tiled (P2)
    - Intra row store: linear (P9)
    - ILDB row store: linear (P10)
- Write streams: (all 64B requests)
    - Row store write for BSD: linear and can avoid partial writes (P3)
    - Row store write for MPR: linear and can avoid partial writes (P4)
    - Intra row store write: linear and can avoid partial writes (P7)
    - ILDB row store write: linear and can avoid partial writes (P8)
    - Final dest writes: tiled and can potentially be partial, two ways to avoid these partials: 1) either write garbage and buffers are aligned or 2) read-modify writes for dribble end of line cases (P11)

MFD Codec-Specific Commands

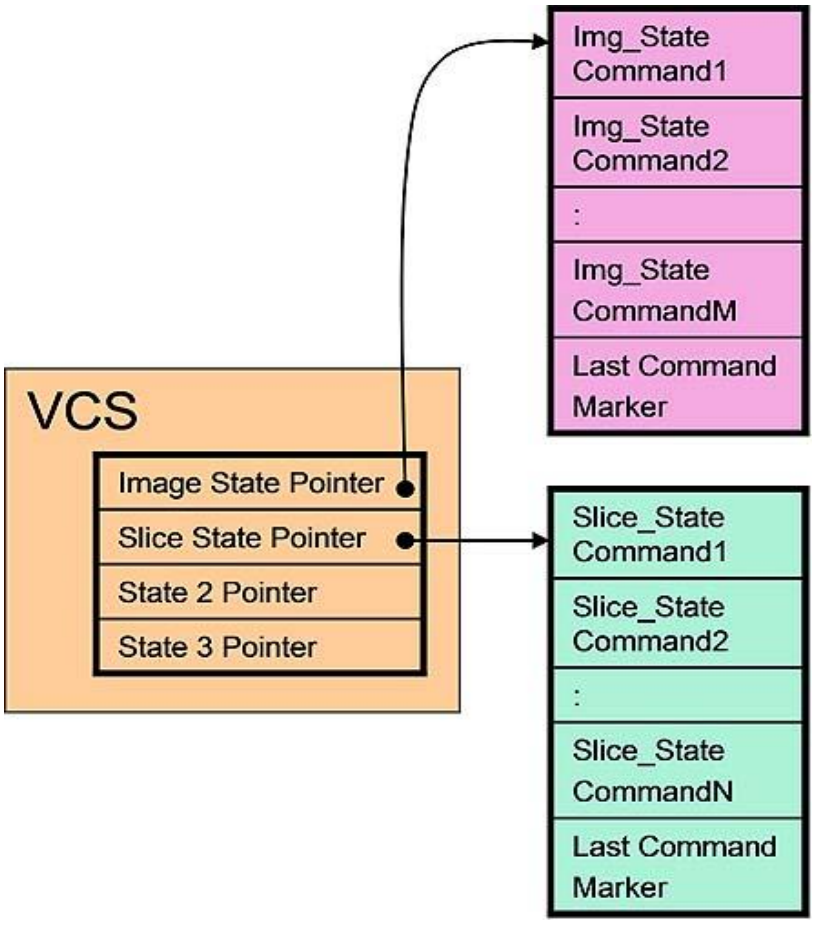
MFD hardware pipeline supports 3 different codec standards: AVC, VC1 and MPEG2. To make the interface flexible, each codec is designed with its own set of commands.

There are two categories of commands for each codec format: one set for VLD mode and one set for IT mode.

MFX State Model

The parallel video engine (PVE) supports two state delivery models: inline state model and indirect state model. For inline state model, the state commands (*_STATE) can be issued in batch buffers or ring buffers directly preceding object commands (*_OBJECT). In the indirect state model, the state commands are not placed in the batch buffers or ring buffers. Instead Indirect State Buffers provide state information (in the form of the above mentioned state commands) for the MFX pipeline. See MFX_STATE_POINTER for more details.

VCS (aka BCS) handles the difference of the two state delivery models. Therefore, the MFX pipeline always sees the state commands in both models. However, MFX hardware supports additional context save/restore of 'dynamic states'. Dynamic states are the internal signals that are persistent. This could be the CABAC context for macroblock encoding.



MFX State Model

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

- In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.

MFX Interruptability Model

MFX encoding and the encoding pipeline do not support interruption. All operations are frame based. Interrupts can only occur between frames; the driver will submit all the states at the beginning of each frame. Any state kept across frames is in MMIO registers that should be read between frames.

Software submits without any knowledge of where the parser head pointer is located. Also there is a non-deterministic amount of time for the new context to reach the command streamer. However, the state model for the MFX engine requires software to know exactly what state the pipeline is in at all times. This introduces cases where a preemption could occur during or after a state change without software ever knowing the state saved out to memory on the context switch.

Also, preemption is only allowed during the last macroblock in a row. Hardware cannot always perform a context switch when the new context is seen by the hardware. To avoid a switch during an invalid macroblock and to keep the state synchronized with software, there are two commands available that are used. MI_ARB_ON_OFF disables and enables preemption while MFX_WAIT ensures the context switch, if needed, preempts during macroblock execution. Below illustrates an example assuming VC1 VLD mode.

Command Ring/Batch	Notes
MI_ARB_ON_OFF = OFF	Disable preemption
S1	Inline or indirect state cmd 1
S2	Inline or indirect state cmd 2
S3	Inline or indirect state cmd 3
XXXX_OBJECT	Slice
MI_ARB_ON_OFF = ON	Enable preemption
MFX_WAIT	Allow preemption to occur while XXXX_OBJECT executes
MI_ARB_ON_OFF = OFF	Since arbitration is off again, state commands are allowed below
S4	Inline or indirect state cmd 4
S5	Inline or indirect state cmd 5
S6	Inline or indirect state cmd 6
XXXX_OBJECT	Slice
MI_ARB_ON_OFF = ON	Enable preemption
MFX_WAIT	Allow preemption to occur while XXXX_OBJECT executes

Command Ring/Batch	Notes
MI_ARB_ON_OFF = OFF	Since arbitration is off again, state commands are allowed below

Note that store DW commands may execute inside the preemption enabling window if needed.

Decoder Input Bitstream Formats

AVC Bitstream Formats – DXVA Short

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process.

AVC Bitstream Formats – DXVA Long

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process. Application will provide the Slice Header Skip Byte count (not including any possible Emulation Prevention Byte).

VC1 Bitstream Formats – Intel Long

Bitstream starts right at the MB layer, with any emulation byte crossing the header and MB layer being removed by application and the data length is adjusted.

MPEG2 Bitstream Formats – DXVA1

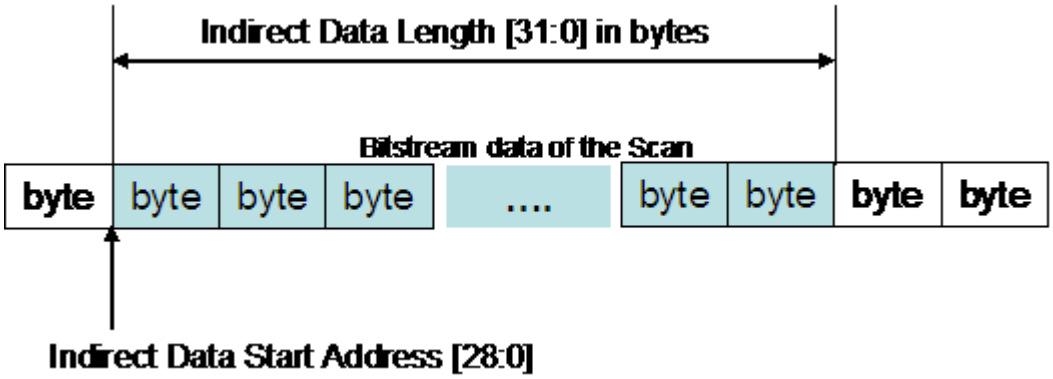
Bitstream buffer starts right at the very first MB data.

JPEG Bitstream Formats – Intel

Bitstream buffer starts right at the very first MCU data of each Scan.

The indirect data start address in MFD_JPEG_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the Scan header. It provides the byte address for the first MCU of the Scan. Different from MFD_MPEG2_BSD_OBJECT command, First MCU Bit Offset does not need to be specified because it is always set to zero.

Indirect data buffer for a Scan



The indirect data length in MFD_JPEG_BSD_OBJECT provides the length in bytes of the bitstream data for the Scan excluding Scan header. It includes the first byte of the first macroblock and the last byte of the last macroblock in the Scan. The Figure illustrates these parameters for a slice data.

Concurrent Multiple Video Stream Decoding Support

The natural place for switching across multiple streams is at the Slice boundary. Each Slice is a self-sustained unit of compressed video data and has no dependency with its neighbors (except for the Deblocking process). In addition, there is no interruptability within a Slice. However, when ILDB is invoked, the processing of some MBs will require neighbor MB information that crosses the Slice boundary. Hence, to limit the buffering requirement, in this version of hardware design, stream switching can only be performed at the picture boundary instead.

MFX Codec Commands Summary

DWord	Bit	Description
0	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** PIPELINE_SELECT GFXPIPE[28:27 = 1h, 26:24 = 1h, 23:16 = 04h] (Single DW, Non-pipelined)
	15:1	**Reserved:** MBZ
	0	**Pipeline Select** 0: 3D pipeline is selected 1: Media pipeline is selected

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	Definition Chapter
VC1 State				
2h	5h	0h	VC1_BSD_PIC_STATE	VC1 BSD
2h	5h	1h	Reserved	n/a
2h	5h	2h	Reserved	n/a
2h	5h	3h	VC1_BSD_BUF_BASE_STATE	VC1 BSD
2h	5h	4h	Reserved	n/a
2h	5h	5h-7h	Reserved	n/a
VC1 Object				
2h	5h	8h	VC1_BSD_OBJECT	VC1 BSD
2h	5h	9h-FFh	Reserved	n/a

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	Definition Chapter
State				
2h	6h	2h-7h	Reserved	N/A
Object				
2h	6h	9h-FFh	Reserved	N/A

Note that it is possible for a command to appear in both IMAGE and SLICE state buffer, e.g. QM_STATE for JPEG can be issued at frame level or scan/slice level.

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	MFX Common	**Common**					
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	No
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	No
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	No
2h	0h	0h	7h	MFX_QM_STATE	MFX	IMAGE/SLICE	No
2h	0h	0h	8h	MFX_FQM_STATE	MFX	IMAGE	No
2h	0h	0h	9h	MFX_DBK_OBJECT	MFX	IMAGE	No
2h	0h	0h	A-1Eh	Reserved	n/a	n/a	No
	MFX Common	**Dec**					
2h	0h	1h	0-8h	Reserved	n/a	n/a	n/a
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	n/a	No
2h	0h	1h	A-1Fh	Reserved	n/a	n/a	n/a
	MFX Common	**Enc**					
2h	0h	2h	0-7Fh	Reserved	n/a	n/a	n/a
2h	0h	2h	8h	MFX_PAK_INSERT_OBJECT	MFX	n/a	No
2h	0h	2h	9h	Reserved	n/a	n/a	n/a
2h	0h	2h	Ah	MFX_STITCH_OBJECT	MFX	n/a	No
2h	0h	2h	B-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Common (State)**					
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	1h	0h	1h	Reserved	n/a	n/a	n/a
2h	1h	0h	2h	MFX_AVC_DIRECTMODE_STATE	MFX	SLICE	n/a
2h	1h	0h	3h	MFX_AVC_SLICE_STATE	MFX	SLICE	n/a
2h	1h	0h	4h	MFX_AVC_REF_IDX_STATE	MFX	SLICE	n/a
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	n/a
2h	1h	0h	9	Reserved	n/a	n/a	n/a
2h	1h	0h	D-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Dec**					
2h	1h	1h	0-5h	Reserved	MFX	n/a	n/a
2h	1h	1h	6h	MFD_AVC_DPB_STATE	MFX	IMAGE	n/a
2h	1h	1h	7h	MFD_AVC_SLICEADDR_OBJECT	MFX	n/a	n/a
2h	1h	1h	8h	MFD_AVC_BSD_OBJECT	MFX	n/a	No
2h	1h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Enc**					
2h	1h	2h	0-8h	Reserved	n/a	n/a	n/a
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	n/a	No
2h	1h	2h	A-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Extension**					
	VC1	**Common (State)**					
2h	2h	0h	0h	Reserved	n/a	n/a	n/a
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	n/a
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	n/a
2h	2h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	VC1	**Dec**					
2h	2h	1h	0h	MFD_VC1_SHORT_PIC_STATE	MFX	IMAGE	n/a
2h	2h	1h	1h	MFD_VC1_LONG_PIC_STATE	MFX	IMAGE	n/a
2h	2h	1h	2-7h	Reserved	n/a	n/a	n/a
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	n/a	No
2h	2h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	VC1	**Enc**					
2h	2h	2h	0-1Fh	Reserved	n/a	n/a	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	MPEG2	**Common (State)**					
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	n/a
2h	3h	0h	1-1Fh	Reserved	n/a	n/a	n/a
	MPEG2	**Dec**					
2h	3h	1h	1-7h	Reserved	n/a	n/a	n/a
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	n/a	No
2h	3h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	MPEG2	**Enc**					
2h	3h	2h	0-2h	Reserved	n/a	n/a	n/a
2h	3h	2h	3h	MFC_MPEG2_PAK_OBJECT			
2h	3h	2h	3-8h	Reserved			
2h	3h	2h	9h	MFC_MPEG2_SLICEGROUP_STATE			
2h	3h	2h	A-1Fh	Reserved			
	VP8	**Common (State)**					
2h	4h	0h	0h	MFX_VP8_PIC_STATE	MFX	IMAGE	n/a
	VP8	**Dec**					
2h	4h	1h	8h	MFD_VP8_BSD_OBJECT	MFX	IMAGE	No
	VP8	**Enc**					
2h	4h	2h		Reserved			
	JPEG	**Common**					
2h	7h	0h	0h	MFX_JPEG_PIC_STATE	MFX	IMAGE	No
2h	7h	0h	1h	Reserved	n/a	n/a	n/a
2h	7h	0h	2h	MFX_JPEG_HUFF_TABLE_STATE	MFX	IMAGE	No
2h	7h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	JPEG	**Common**					
2h	7h	0h	0h	MFX_JPEG_PIC_STATE	MFX	IMAGE	No
2h	7h	0h	1h	Reserved	n/a	n/a	n/a
2h	7h	0h	2h	MFX_JPEG_HUFF_TABLE_STATE	MFX	IMAGE	No
2h	7h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	JPEG	**Dec**					
2h	7h	1h	1-7h	Reserved	MFX	n/a	n/a
2h	7h	1h	8h	MFD_JPEG_BSD_OBJECT	MFX	MCU	No
2h	7h	1h	9-1Fh	Reserved	MFX	n/a	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	JPEG	Enc					
2h	7h	2h	0-1Fh	Reserved	MFX	n/a	n/a

MMIO Space Registers

Range Start	Range End	Unit owner
00002000	00002FFF	Render/Generic Media Engine
00004000	00004FFF	Render/Generic Media Graphics Memory Arbiter
00005000	0000517F	
00006000	00007FFF	Reserved
00012000	000123FF	MFX Control Engine (Video Command Streamer)
00012400	00012FFF	Media Units (VIN unit)
00014000	00014FFF	MFX Memory Arbiter
00022000	00022FFF	Blitter Engine
00024000	00024FFF	Blitter Memory Arbiter
00030000	0003FFFF	Reserved
00100000	00107FFF	Fence Registers
00140000	0017FFFF	MCHBAR (SA)

Memory Interface Command Map

04h Opcode (28:23)	MI_FLUSH

MFX Decoder Commands Sequence

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition

Examples for AVC

The following gives a sample command sequence programmed by a driver

a) For Intel or DXVA2 AVC Long Slice Bitstream Format

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_QM_STATE

VLD mode: MFX_AVC_PICID_STATE

MFX_AVC_IMG_STATE

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_REF_IDX_STATE

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_SLICE_STATE

VLD mode: MFD_AVC_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 AVC Short Slice Bitstream Format (for VLD mode only)

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFD_AVC_DPB_STATE

VLD mode: MFX_AVC_PICID_STATE

MFX_AVC_IMG_STATE

MFX_QM_STATE

MFX_AVC_DIRECTMODE_STATE

VLD mode : MFD_AVC_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

VLD mode : MFD_AVC_BSD_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

... repeat these four commands N-1 times for a N-slice picture

VLD mode: MFD_AVC_BSD_OBJECT (for the last slice of the picture)

MI_FLUSH

Examples for VC1

The following gives a sample command sequence programmed by a driver

a) For Intel Proprietary Long Bitstream Format

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_LONG_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only)

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_SHORT_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

MI_FLUSH

c) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only), and field pair picture

Batch buffer for top-field

states....

Slice_objs...

MI_flush

store register immediate (if VC1 short format with interlaced field pic)

MI_flush

Batch buffer for bottom field

load register immediate (if VC1 short format with interlaced field pic)

MI_flush

states....

Slice_objs...

MI_flush

Examples for JPEG

The following gives a sample command sequence programmed by a driver

Programmed once at the start of decoding

MFX_PIPE_MODE_SELECT

MFX_PIPE_SURFACE_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_JPEG_PIC_STATE

Programmed at the start of Frame or Scan (These commands can be sent multiple times either before MFX_JPEG_PIC_STATE or before MFD_JPEG_BSD_OBJECT)

MFX_JPEG_HUFF_TABLE

MFX_QM_STATE

Programmed per Scan (These commands can be sent multiple times depending on each bit stream)

MFD_JPEG_ BSD_OBJECT

MI_FLUSH

MFX Pipe Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_STATE_POINTER

MFX_PIPE_MODE_SELECT

The Encoder Pipeline Modes of Operation (Per Frame):

1. PAK Mode: VCS-command driven, setup by driver. Like the IT mode of decoder, it is executed on a per-MB basis. Hence, each PAK Object command corresponds to coding of only one MB.
    a. Normal Mode (including transcoding): receive per-MB control and data (MV, mb_type, cbp, etc.). It generates the output compressed bitstream as well as the reconstructed reference pictures, one MB at a time, for later use.
    b. Encoder StreamOut Mode: to provide per-MB, per-Slice and per-Frame coding result and information (statistics) to the Host, Video Preprocessing Unit and ENC Unit to enhance their operations.

The Decoder Pipeline Modes of Operation (Per Frame):

1. VLD Mode: The output from the BSD (weight&offset/coeff/motion vectors record) can be sent in part (as specified) and to the remaining fixed function hardware pipeline to complete the decoding processing. The driver specifies through MFD commands of what to send out from the BSD unit and where to send the BSD output.
    a. For transcoding (including transrating and transcaling), part of the BSD output (a series of per-MB record) can be sent to memory for further processing to encode into a difference output format. This function is named as StreamOut. When StreamOut is active, not all MB information needs to be sent, only MVs and selective MB coding information.
2. IT Mode: In this mode, the BSD is not invoked. Instead host performs all the bitstream decoding and parsing; and the result are saved into memory in a specific per-MB record format. The MFD Engine VCS reads in these records one at time and finish the rest of the decoding (IT, MC, IntraPred and ILDB).

MB information is organized into two indirect data buffers, one for MVs and one for residue coefficients. As such, two indirect base address pointers are defined.

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_PAK_INSERT_OBJECT

MFX_STITCH_OBJECT

MFX_QM_STATE

Bits	31:24	23:16	15:8	7:0
Dword 1	QuantMatrix[0][3]	QuantMatrix[0][2]	QuantMatrix[0][1]	QuantMatrix[0][0]
Dword 2	QuantMatrix[0][7]	QuantMatrix[0][6]	QuantMatrix[0][5]	QuantMatrix[0][4]
Dword 3	QuantMatrix[1][3]	QuantMatrix[1][2]	QuantMatrix[1][1]	QuantMatrix[1][0]
...	...	...	...	...
Dword 16	QuantMatrix[7][7]	QuantMatrix[7][6]	QuantMatrix[7][5]	QuantMatrix[7][4]

MFX_FQM_STATE

This is a frame-level state. Reciprocal Scaling Lists are always sent from the driver regardless whether they are specified by an application or the default/flat lists are being used. This is done to save the ROM (to store the default matrices) inside the PAK Subsystem. Hence, the driver is responsible for determining the final set of scaling lists to be used for encoding the current slice, based on the AVC Spec (Fall-Back Rules A and B). For encoding, there is no need to send the qm_list_flags[i], i=0 to7 and qm_present_flag to the PAK, since Scaling Lists syntax elements are encoded above Slice Data Layer.

FQM Reciprocal Scaling Lists elements are 16-bit each, conceptually equal to 1/ScaleValue. QM matrix elements are 8-bit each, equal to ScaleValue. However, in AVC spec., the Reciprocal Scaling Lists elements are not exactly equal to one-over of the corresponding Scaling Lists elements. The numbers are adjusted to simplify hardware implementation.

For all the description below, a scaling list set contains 6 4x4 scaling lists (or forward scaling lists) and 2 8x8 scaling lists (or forward scaling lists).

In MFX PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order as shown in MFX_AVC_QM_STATE. But the Forward Q scaling lists are sent in transport form, i.e. column-wise raster order (column-by-column) to simplify the H/W.

Precisely, if the reciprocal forward scaling matrix is F[4][4], then the 16 word of the matrix will be set as the following:

For JPEG encoder, 16-bit precision is used for each element 1/QM matrix. The 32 DWords are used for 64 QM elements with the following data structure:

	Bits 15:0	Bits 31:16
DWord1	1/QM[0][0]	1/QM[1][0]
DWord2	1/QM[2][0]	1/QM[3][0]
DWord3	1/QM[4][0]	1/QM[5][0]
DWord4	1/QM[6][0]	1/QM[7][0]
DWord5	1/QM[0][1]	1/QM[1][1]
DWord6	1/QM[2][1]	1/QM[3][1]
DWord7	1/QM[4][1]	1/QM[5][1]
DWord8	1/QM[6][1]	1/QM[7][1]
...		
DWord31	1/QM[4][7]	1/QM[5][7]

	Bits 15:0	Bits 31:16
DWord32	1/QM[6][7]	1/QM[7][7]

Bitplane Buffer

Bitplane coding is used in seven different cases in VC-1, although not all the seven syntax elements are present in the same picture header at the same time. The following list shows which syntax elements are coded as bitplanes in each picture header:

Progressive I and BI picture headers in AP: ACPRED, OVERFLAGS

Field interlace I and BI picture headers in AP: ACPRED, OVERFLAGS

Frame interlace I and BI picture headers in AP: FIELDTX, ACPRED, OVERFLAGS

Frame interlace P picture headers in AP: SKIPMB

Progressive P picture headers in SP and MP: MVTYPEMB, SKIPMB

Progressive P picture headers in AP: MVTYPEMB, SKIPMB

Field interlace B picture headers in AP: FORWARDMB

Frame interlace B picture headers in AP: DIRECTMB, SKIPMB

Progressive B picture headers in AP: DIRECTMB, SKIPMB

Progressive B picture headers in MP: DIRECTMB, SKIPMB

There are also seven different modes of coding the bitplane information. Except when the bitplane is coded in raw mode, the bitplane is decoded by the host and provided to the hardware in the bitplane buffer.

Since at most three bitplanes are encoded in any picture header, instead of using a complete byte for signaling the values of all the seven possible bitplanes for each MB, a more efficient approach is used with each byte divided in two nibbles and each nibble carries the data of up to four bitplanes for one MB.

PictureType	Bits 3, 7	Bit 2, 6	Bits 1, 5	Bits 0, 4
I or BI	0	OVERFLAGS	ACPRED	FIELDTX
P	0	MVTYPEMB	SKIPMB	0
B	0	FORWARDMB	SKIPMB	DIRECTMB

The bytes containing the above defined nibbles are stored in the bitplane buffer in raster scan order. The bitplane buffer is a linear buffer with a buffer pitch (as defined by Bitplane Buffer Pitch field in VC1_BSD_PIC_STATE command). When the number of macroblock in a row is odd, the last byte of the row containing the last macroblock in bits 0-3. The first macroblock of the next row starts at the next pitch offset from the first macroblock of the current row.

The bitplane buffer structure must be sent once per picture only if there is one or more syntax elements coded as bitplanes in the picture header.

Video Codecs

The following sections contain the various registers for video codec support. Specifically, the codec types supported are:

Supported Codec Types
Advanced Video Coding (AVC)/ H.264/MPEG-4 Part 10 (MVC)
MPEG-2 (H.222/H.262) — Used in Digital Video Broadcast and DVDs
VC1 — SMPTE 421M, known informally as VC-1
VP8 — Video compression format
JPEG and MJPEG — A video format in which video gram or interlaced field of a digital video sequence is compressed separately as a JPEG image
Other Codec Functions

Internal Media Rowstore table – An internal Media Rowstore Storage is added to reduce memory read/write to save power. If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

AVC/VC1/MPEG2/JPEG/VP8 Decoder/Encoder:

[BSD is bitstream decoder rowstore; MPR is Motion Prediction rowstore; IP is Intra Prediction rowstore; VLF is loop filter rowstore]

Codec	Mode	Frame Width	BSD	MPR	IP	VLF	VDE	BSD Addr	MPR Addr	IP Addr	VLF ADDR	VDE ADDR
AVC Dec	Frame/Field	< 2k	Y	Y	Y	Y	N	0	128	256	384	N/A
		2k -> 3k	Y	Y	Y	N	N	0	192	384	N/A	N/A
		3k -> 4k	Y	Y	N	N	N	0	256	N/A	N/A	N/A
	Mbaff	< 2k	Y	Y	Y	N	N	0	256	512	N/A	N/A
		2k -> 4k	Y	N	N	N	N	0	N/A	N/A	N/A	N/A
AVC Enc	Frame/Field	< 2k	Y	N	Y	Y	N	0	N/A	256	384	N/A
		2k -> 3k	Y	N	Y	N	N	0	N/A	384	N/A	N/A
		3k -> 4k	Y	N	Y	N	N	0	N/A	256	N/A	N/A
	Mbaff	< 2k	Y	N	Y	N	N	0	N/A	512	N/A	N/A
		2k -> 4k	Y	N	N	N	N	0	N/A	N/A	N/A	N/A
JPEG Dec/Enc			N	N	N	N	N	N/A	N/A	N/A	N/A	N/A
VP8 Dec	Frame	< 2k	Y	N	Y	Y	N	0	N/A	256	384	N/A
		2k -> 3k	Y	N	Y	N	N	0	N/A	384	N/A	N/A
		3k -> 4k	Y	N	Y	N	N	0	N/A	256	N/A	N/A
VP8 Enc	Frame		N	N	N	N	N	N/A	N/A	N/A	N/A	N/A
MPEG2			N	N	N	N	N	N/A	N/A	N/A	N/A	N/A

Codec	Mode	Frame Width	BSD	MPR	IP	VLF	VDE	BSD Addr	MPR Addr	IP Addr	VLF ADDR	VDE ADDR
VC1 Dec			Y	N	N	N	N	N/A	N/A	N/A	N/A	N/A

AVC (H.264)

AVC Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_AVC_IMG_STATE

A new command is added to support MPEG transport stream encapsulation of AVC bitstream in Encoder mode. This command should be used only when MPEG transport stream is needed.

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_SLICE_STATE

MFX_AVC_REF_IDX_STATE

MFX_AVC_WEIGHTOFFSET_STATE

AVC Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_AVC_DPB_STATE

NOTE modified from DXVA2 – The values in RefFrameList and UsedForReference_Flag are the primary means by which the H/W can determine whether the corresponding entries in RefFrameList, POCList, LTSTFrameNumList, and Non-ExistingFrame_Flag should be considered valid for use in the decoding process of the current picture or not. When RefFrameList[i] is marked to be invalid, the values of POCList[i][0], POCList[i][1], LTSTFrameNumList[i], UsedForReference_Flag[i], and Non-ExistingFrame_Flag[i] must all be equal to 0. When UsedForReference_Flag[i] = 0, the value of RefFrameList[i] must be marked invalid.

MFD_AVC_SLICEADDR

MFD_AVC_BSD_OBJECT

Inline Data Description for MFD_AVC_BSD_Object

MFD_AVC_PICID_STATE

NOTE 1: In AVC short format, PictureIDList has one-to-one corresponding to LongTermFrame_Flag list, Non-ExistingFrame_flag list, UsedForReference_Flag list, FrameNumList list in MFD_AVC_DPB_STATE.

NOTE 2: PictureIDList is only used to identify reference picture across frames. Hardware will convert the picture in the RefFrameList to PictureID before writing out DMV data and convert back to RefFrameList Index after reading out DMV data. The reference pictures and their orders in the RefFrameList can be changed across frames.

Session Decoder StreamOut Data Structure

When StreamOut is enabled, per MB intermediated decoded data (MVs, mb_type, MB qp, etc.) are sent to the memory in a fixed record format (and of fixed size). The per-MB records must be written in a strict raster order and with no gap (i.e. every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

A StreamOut Data record format is detailed as follows:

DWord	Bit	Description
	23	**Reserved MBZ**
	22-20	**EdgeFilterFlag** (AVC) / **OverlapSmoothFilter** (VC1)
	19:17	**CodedPatternDC** (for AVC only, **111b** for others) The field indicates whether DC coefficients are sent. 1 bit each for Y, U and V.
	16	**Reserved MBZ**
	15	**Transform8x8Flag** When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 transform. The transform_szie_8x8_flag syntax element, if present in the output bitstream, is the same as this field. However, whether transform_szie_8x8_flag is present or not in the output bitstream depends on several conditions: This field is only allowed to be set to 1 for two conditions: It must be 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8 It may be 1 if **IntraMbFlag** = INTER and there is no sub partition size less than 8x8 Otherwise, this field must be set to 0. 0: 4x4 integer transform 1: 8x8 integer transform
	14	**MbFieldFlag** This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. Same as the mb_field_decoding_flag syntax element in AVC spec. 0 = Frame macroblock 1 = Field macroblock

DWord	Bit	Description
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) **1: INTRA (intra macroblock)**
	12:8	**MbType5Bits** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows AVC encoding for inter and intra macroblocks.
	7	**MbPolarity** FieldMB Polarity - vctrl_vld_top_field AVC
	6	**Reserved MBZ**
	5:4	**IntraMbMode** This field is provided to carry information partially overlapped with MbType. This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware.
	3	**Reserved MBZ**
	2	**MbSkipFlag** Reserved MBZ (DXVA Encoder). HW (VDSunit) doesn't have skip MB info. It sets to 1 if any of the sub-blocks is inter, uses predicted MVs, and skips sending MVs explicitly in the code stream. Currently H/W can provide this flag and is defaulted to 0 always.
	1:0	**InterMbMode** This field is provided to carry redundant information as that in MbType. It also carries additional information such as skip. This field is only valid if **IntraMbFlag** =INTER, otherwise, it is ignored by hardware.
1	31:16	**MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
	15:0	**MbXCnt (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
2	31	**Conceal MB Flag.** This field specifies if the current MB is a conceal MB, use in AVC/VC1/MPEG2 mode.
	30	**Last MB of the Slice Flag.** This field indicate the current MB is a last MB of the slice. Use in AVC/VC1/MPEG2 mode.
	29:24	**Reserved**
	23:20	**CbpAcV**

DWord	Bit	Description
		0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).
	19:16	**CbpAcU** 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).
	15:0	**CbpAcY** 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding). Bit15=Y0Sub0, Bit0=Y3Sub3
3	31:28 AVC	**Skip8x8Pattern** (AVC) This field indicates whether each of the four 8x8 sub macroblocks is using the predicted MVs and will not be explicitly coded in the bitstream (the sub macroblock will be coded as direct mode). It contains four 1-bit subfields, corresponding to the 4 sub macroblocks in sequential order. The whole macroblock may be actually coded as B_Direct_16x16 or B_Skip, according to the macroblock type conversion rules described in a later sub section. This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock. 0 in a bit – Corresponding MVs are sent in the bitstream 1 in a bit – Corresponding MVs are not sent in the bitstream
	27:25	**Reserved**
	24:16	**NzCoefCountMB** – all coded coefficients input including AC/DC blocks in current MB. Range 0 to 384 (9 bits)
	15:8	**MbClock16** – MB compute clocks in 16-clock unit.
	7	**mbz** (AVC) **/ QScaleType** (MPEG2)
	6:0	**QpPrimeY** (AVC) **/ QScaleCode** (MPEG2) The luma quantization index. This is the per-MB QP value specified for the current MB.
4 to 6	31:0 Each	For intra macroblocks, definition of these fields are specified in 1 For inter macroblocks, definition of these fields are specified in 2
7	31:24	**Reserved**

DWord	Bit	Description
	23:20	**MvFieldSelect** (Ref polarity top or bottom bits) for VC1 and MPEG2 vcp_vds_mvdataR[162:159] VC1 vmd_vds_mt_vert_fld_selR[3:0] MPEG2
	19:12	**Reserved**
	11:10	**SubBlockCodeType V** (If 8x8, 8x4, 4x8, 4x4 type)
	9:8	**SubBlockCodeType U** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	7:6	**SubBlockCodeType Y3** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	5:4	**SubBlockCodeType Y2** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	3:2	**SubBlockCodeType Y1** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	1:0	**SubBlockCodeType Y0** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
Inter Cases:		
8	31:16	**MvFwd[0].y** – y-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock
	15:0	**MvFwd[0].x** – x-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock
9	31:0	**MvBck[0]** – the backward motion vector of the 1st 8x8 or 1st 4x4 subblock
10	31:0	**MvFwd[1]** – the forward motion vector of the 2nd 8x8 or 4th 4x4 subblock
11	31:0	**MvBck[1]** – the backward motion vector of the 2nd 8x8 or 4th 4x4 subblock
12	31:0	**MvFwd[2]** – the forward motion vector of the 3rd 8x8 or 8th 4x4 subblock
13	31:0	**MvBck[2]** – the backward motion vector of the 3rd 8x8 or 8th 4x4 subblock
14	31:0	**MvFwd[3]** – the forward motion vector of the 4th 8x8 or 12th 4x4 subblock
15	31:0	**MvBck[3]** – the backward motion vector of the 4th 8x8 or 12th 4x4 subblock
Intra Cases:		
8 to 15	31:0	**Reserved MBZ**

The inline data content of Dwords 4 to 6 is defined either for intra prediction or for inter prediction, but not both.

Inline data subfields for an Intra Macroblock

DWord	Bit	Description
4	31:16	**LumaIndraPredModes[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.

DWord	Bit	Description
		AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block (Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes. 4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
5 AVC INTRA	31:16	**LumaIndraPredModes[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment later in this section. VC1: MBZ. MPEG2: MBZ.
6	31:8	Reserved (Reserved for encoder turbo mode **IntraResidueDataSize**, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)
	7:0	**MbIntraStruct** The IntraPredAvailFlags[4:0] have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbors position around the current MB or MB pair (in MBAFF mode). 1 – IntraPredAvailFlagX, indicates the values of samples of neighbor X can be used in intra prediction for the current MB. 0 – IntraPredAvailFlagX, indicates the values of samples of neighbor X is not available for intra

DWord	Bit	Description
		prediction of the current MB. IntraPredAvailFlag-A and -E can only be different from each other when constrained_intra_pred_flag is equal to 1 and mb_field_decoding_flag is equal to 1 and the value of the mb_field_decoding_flag for the macroblock pair to the left of the current macroblock is equal to 0 (which can only occur when MbaffFrameFlag is equal to 1). IntraPredAvailFlag-F is used only if ○ it is in MBAFF mode, i.e. **MbaffFrameFlag** = 1, ○ the current macroblock is of frame type**,** i.e. **MbFieldFag** = 0, and ○ the current macroblock type is Intra8x8, that is, **IntraMbFlag** = INTRA, **IntraMbMode** = INTRA_8x8, and **Transform8x8Flag** = 1. In any other cases IntraPredAvailFlag-A shall be used instead.

Bits	IntraPredAvailFlags[4:0] Definition
7	**IntraPredAvailFlagF – F** (Left 8th row (-1,7) neighbor)
6	**IntraPredAvailFlagA – A** (Left neighbor top half)
5	**IntraPredAvailFlagE – E** (Left neighbor bottom half)
4	**IntraPredAvailFlagB – B** (Top neighbor)
3	**IntraPredAvailFlagC – C** (Top right neighbor)
2	**IntraPredAvailFlagD – D** (Top left corner neighbor)
1:0	**ChromaIntraPredMode –** 2 bits to specify 1 of 4 chroma intra prediction mode, see the table in later section.

Inline data subfields for an Inter Macroblock

DWord	Bit	Description
4	31:24	Reserved: MBZ (DXVA Decoder)
	23:16	Reserved: MBZ (DXVA Decoder)
	15:8	**SubMbPredModes**[bit 7:0] (Sub Macroblock Prediction Mode) This field describes the prediction mode of the sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA) This field is derived from MbType for a non-BP_8x8 inter macroblock, and carries redundant

DWord	Bit	Description
		information as MbType) Bits [1:0]: SubMbPredMode[0] – for 8x8 Block 0 Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1 Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2 Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3 Blocks of the MB is numbered as follows : 0 1 2 3 Each 2-bit value [1:0] is defined as : 00 – Pred_L0 01 – Pred_L1 10 – BiPred **For VC1:** Bits [1:0]: "00"= Y0 Forward only, "01"= Y0 Backward only, "10"= Y0 Bi direction Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1 Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2 Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3
	7:0	**SubMbShape**[bit 7:0] (Sub Macroblock Shape) This field describes the sub-block partitioning of each sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA) This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as MbType). Bits [1:0]: SubMbShape[0] – for 8x8 Block 0 Bits [3:2]: SubMbShape[1] – for 8x8 Block 1 Bits [5:4]: SubMbShape[2] – for 8x8 Block 2 Bits [7:6]: SubMbShape[3] – for 8x8 Block 3 Blocks of the MB is numbered as follows : 0 1 2 3

DWord	Bit	Description
		Each 2-bit value [1:0] is defined as : 00 – SubMbPartWidth=8, SubMbPartHeight=8 01 – SubMbPartWidth=8, SubMbPartHeight=4 10 – SubMbPartWidth=4, SubMbPartHeight=8 11 – SubMbPartWidth=4, SubMbPartHeight=4 For VC-1, This field indicates the transformation types used for luma components, 2 bits for each 8x8.
5	31:24	**Frame Store ID L0[3]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	23:16	**Frame Store ID L0[2]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	15:8	**Frame Store ID L0[1]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format.

DWord	Bit	Description
		0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation).
	7:0	**Frame Store ID L0[0]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
6	31:24	**Frame Store ID L1[3]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	23:16	**Frame Store ID L1[2]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel

DWord	Bit	Description
		implementation)
	15:8	**Frame Store ID L1[1]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	7:0	**Frame Store ID L1[0]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)

AVC Encoder PAK Commands

Each PAK Commands is composed of a command op-code DW and one or more command data DWs (inline data). The size of each command is specified as part of the op-code DW. Most of the commands have fixed size, except some are allowed to be of variable length.

There is an inherent order of executing MFC PAK commands that driver must follow.

MFC_AVC_PAK_OBJECT

PAK Object Inline Data Description

The Inline Data includes all the required MB encoding states, constitute part of the Slice Data syntax elements, MB Header syntax elements and their derivatives. It provides information for the following operations:

1. Forward and Inverse Transform
2. Forward and Inverse Quantization
3. Advanced Rate Control (QRC)
4. MB Parameter Construction (MPC)
5. CABAC/CAVLC encoding
6. Bit stream packing
7. Intra and inter-Prediction decoding loop
8. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_AVC_PAK_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_AVC_PAK_OBJECT command.

The inline data has been designed to match the DXVA 2.0, with the exception of the starting byte (DW0:0-7) and the ending dword (DW7:0-31).

The Deblocker Filter Control flags (FilterInternalEdgesFlag, FilterTopMbEdgeFlag and FilterLeftMbEdgesFlag) are generated by H/W, which are depending on MbaffFrameFlag, CurrMbAddr, PicWidthInMbs and disable_deblocking_filter_idc states.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

DWord	Bit	Description
	30	**Reserved: MBZ**
	23	**Reserved: MBZ**
	19	**CbpDcY.** This field specifies if the Luma DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process. 1 – the 4x4 DC-only Luma sub-block of the Intra16x16 coded MB is present; it is still possible that all DC coefficients are zero. 0 – no 4x4 DC-only Luma sub-block is present; either not in Intra16x16 MB mode or all DC coefficients are zero. **Programming Note** **Context:** PAK Object Inline Data Description - CbpDcY When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.
	18	**CbpDcU.** This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process. 1 – the 2x2 DC-only Chroma Cb sub-block of all coded MB (any type) is present; it is still possible that all DC coefficients are zero. 0 – no 2x2 DC-only Chroma Cb sub-block is present; all DC coefficients are zero. **Programming Note** **Context:** PAK Object Inline Data Description When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td>17</td><td>CbpDcV. This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process.

1 – the 2x2 DC-only Chroma Cr sub-block of all coded MB (any type) is present; it is still possible that all DC coefficients are zero.

0 – no 2x2 DC-only Chroma Cr sub-block is present; all DC coefficients are zero.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>PAK Object Inline Data Description</td></tr><tr><td colspan="2">When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1’s.</td></tr></table></td></tr>
<tr><td></td><td>16</td><td>Reserved: MBZ
(reserved for future use as ExternalResidBufFlag for turbo mode)</td></tr>
<tr><td></td><td>15</td><td>Transform8x8Flag

This field indicates that 8x8 transform is used for the macroblock.

When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 transform. The transform_size_8x8_flag syntax element, if present in the output bitstream, is the same as this field. However, whether transform_szie_8x8_flag is present or not in the output bitstream depends on several other conditions.

This field is only allowed to be set to 1 for two conditions:

It must be 1 if IntraMbFlag = INTRA and IntraMbMode = INTRA_8x8

It may be 1 if IntraMbFlag = INTER and there is no sub partition size less than 8x8

Otherwise, this field must be set to 0.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>PAK Inline Object Data Description</td></tr><tr><td colspan="2">When AvcImgState EntropyCodingFlag is 1(CABAC), this field cannot be 1.</td></tr></table>
0: 4x4 integer transform

1: 8x8 integer transform</td></tr>
</table>

DWord	Bit	Description
	14	**FieldMbFlag** This field specifies the field polarity of the current macroblock, as the mb_field_decoding_flag syntax element in AVC spec. This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. It is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. 0 = Frame macroblock 1 = Field macroblock
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	**MbType5Bits** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows an unified encoding for inter and intra macroblocks according to AVC Spec.
	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. Within an MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in a MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0. Within a field picture, this field is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. It is a constant for the whole field picture. This field is reserved and MBZ for a progressive frame picture. 0 = Current macroblock is a field macroblock from the **top** field 1 = Current macroblock is a field macroblock from the **bottom** field *Programming Note: Here bits [26:24] (MbAffFieldFlag and FiedlMbPolarityFlag) match with bits [10:8] of the Media Block Read message descriptor, simplifying the programming for message generation, as when MbAffFieldFlag is "1", kernels need to override the original "frame" surface state set for MBAFF frame picture.*
	6	MB Reserved: Inter MB converted to IPCM. This field is for HW purposes only. SW should not use it.

DWord	Bit	Description
	5:4	**IntraMbMode** This field is provided to carry information partially overlapped with MbType. This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware..
	3	Reserved: MBZ
	2	**SkipMbFlag** By setting it to 1, this field forces an inter macroblock to be encoded as a skipped macroblock. It is equivalent to mb_skip_flag in AVS spec, indicating that a macroblock is inferred as a P_Skip (or B_Skip) in a P Slice (or B Slice). Hardware honors input MVs for motion prediction and forces CBP to zero. By setting it to 0, an inter macroblock will be coded as a normal inter macroblock. The macroblock may still be coded as a skipped macroblock, according to the macroblock type conversion rules described in the later sub sections. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. For intra MB (bit 13 of this DW set to one), this bit must be set to zero. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock
	1:0	**InterMbMode** This field is provided to carry redundant information as that encoded in MbType. This field is only valid if **IntraMbFlag** =0, otherwise, it is ignored by hardware.
4	15:8	MbYCnt (Vertical Origin). This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
	7:0	MbXCnt (Horizontal Origin). This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
5	31:16	Cbp4x4V (Coded Block Pattern Cr) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cr sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCr bit assignment is cbpCr bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values

DWord	Bit	Description
		are zero), or force to zero for PAK. 1 in a bit - indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero - bad coding). For monochrome, this field is ignored. For 4.2.2, [23:16] for U(Cb), and [31:24] ignored. For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block. **Programming Note** **Context:** PAK Object Inline Data Description When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.
5	15:0	Cbp4x4U (Coded Block Pattern Cb) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cb sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCb bit assignment is cbpCb bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 1 in a bit - indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero - bad coding). For monochrome, this field is ignored. For 4.2.2, [7:0] for U(Cb), and [15:8] ignored. For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block. **Programming Note** **Context:** PAK Object Inline Data Description When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.
6	31:28	**Skip8x8Pattern** This field indicates whether each of the four 8x8 sub macroblocks is using the predicted MVs and will not be explicitly coded in the bitstream (the sub macroblock will be coded as direct mode). It contains four 1-bit subfields, corresponding to the 4 sub macroblocks in sequential order. The whole macroblock may be actually coded as B_Direct_16x16 or B_Skip, according to the macroblock

DWord	Bit	Description
		type conversion rules described in a later sub section. This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock. 0 in a bit – Corresponding MVs are sent in the bitstream 1 in a bit – Corresponding MVs are not sent in the bitstream
	27	**EnableCoeffClamp** 1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization 0 = no clamping
	26	**LastMbFlag** 1 – the current MB is the last MB in the current Slice 0 – the current MB is not the last MB in the current Slice - Reserved MBZ.
	25	**SkipMbConvDisable** This is a per-MB level control to enable and disable skip conversion. This field is ORed with SkipConvDisable field. This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Section *Macroblock Type Conversion Rules* 0 - Enable skip type conversion for the current macroblock 1 - Disable skip type conversion for the current macroblock
	24	Reserved MBZ.
	23:16	**Reserved. Ignored by HW, this field will be re-derived internally**. (was **QpPrimeV.** For 8-bit pixel data, QpCr is the same as QpPrimeCr, and it takes on a value in the range of 0 to 51, positive integer.)
	15:8	**Reserved. Ignored by HW, this field will be re-derived internally**. (Was **QpPrimeU.** For 8-bit pixel data, QpCb is the same as QpPrimeCb, and it takes on a value in the range of 0 to 51, positive integer.)
	7:0	**QpPrimeY** This is the per-MB QP value specified for the current MB. For 8-bit pixel data, QpY is the same as QpPrimeY, and it takes on a value in the range of 0 to 51, positive integer. **Programming Note** **Context:** PAK Object Inline Data Description This value may differ from the actual codes, when HW QRC is on

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>7 .. 9</td><td>31:0
Each</td><td>For intra macroblocks, definition of these fields are specified in PAK Object Inline Data Description
For inter macroblocks, definition of these fields are specified in PAK Object Inline Data Description</td></tr>
<tr><td rowspan="4">10</td><td>31:24</td><td>MaxSizeInWord
PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.</td></tr>
<tr><td>23:16</td><td>TargetSizeInWord
PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients.</td></tr>
<tr><td>15:0</td><td>Reserved: MBZ</td></tr>
<tr><td>15:0</td><td>Lambda_Or_RoundingOffset
When TQEnb=1, this 16-bit unsigned value multiplied by 2 is used as a lambda for the rate-distortion cost estimation in Trellis quantization (TQ). If the upper 4 bits are all set to 1 (0xFXXX), TQ is disabled and the regular quantizer is used. Thus, the valid range is 0~0xEFFF. When TQ is enabled per MB, the TQR in MFC_AVC_IMG_STATE is used for rounding quantization coefficients.
When TQEnb=0 or the upper 4 bits are all set to 1, the lower 4-bit value indicates the rounding precision (offset) for the regular quantizer. The values ranging 0001b ~ 0111 are reserved.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0000b</td><td>RoundInterEnable, RoundInter, RoundIntraEnable, and RoundIntra defined in MFC_AVC_SLICE_STATE are used as rounding precision.</td></tr>
<tr><td>1000b</td><td>+1/16</td></tr>
<tr><td>1001b</td><td>+2/16</td></tr>
<tr><td>1010b</td><td>+3/16</td></tr>
<tr><td>1011b</td><td>+4/16</td></tr>
<tr><td>1100b</td><td>+5/16</td></tr>
<tr><td>1101b</td><td>+6/16</td></tr>
<tr><td>1110b</td><td>+7/16</td></tr>
<tr><td>1111b</td><td>+8/16</td></tr>
</table>
Format: U16</td></tr>
</table>

Inline data for LumaIntraMode

	0 or 1	0	0	1	1
ExtendedForm	Intra4x4	Intra8x8	Intra16x16	Intra8x8	Intra16x16
DW4 – 31:28	Block 7	-	-	-	Block 0
DW4 – 27:24	Block 6	-	-	-	Block 0
DW4 – 23:20	Block 5	-	-	-	Block 0
DW4 – 19:16	Block 4	-	-	-	Block 0
DW4– 15:12	Block 3	-	-	-	Block 0
DW4 – 11:8	Block 2	-	-	-	Block 0
DW4 – 7:4	Block 1	-	-	-	Block 0
DW4 – 3:0	Block 0	-	-	-	Block 0
DW5 – 31:28	Block 15	-	-	-	Block 0
DW5 – 27:24	Block 14	-	-	-	Block 0
DW5 – 23:20	Block 13	-	-	-	Block 0
DW5 – 19:16	Block 12	-	-	-	Block 0
DW5 – 15:12	Block 11	-	-	-	Block 0
DW5– 11:8	Block 10	-	-	-	Block 0
DW5 – 7:4	Block 9	-	-	-	Block 0
DW5 – 3:0	Block 8	-	-	-	Block 0

vctrl_pred_mode[63:0]	(vctrl_it_lumaintrapredmode3[15:0] & vctrl_it_lumaintrapredmode2[15:0] & vctrl_it_lumaintrapredmode1[15:0] & vctrl_it_lumaintrapredmode0[15:0]) : vctrl_pred_mode_noextend[63:0]
vctrl_pred_mode_noextend[63:0]	(vctrl_INTRA_vld_16x16mode & vctrl_it_Transform8x8Flag) ? vctrl_pred_mode_noextend_4x4[63:0] : vctrl_pred_mode_noextend_16x16[63:0] : vctrl_pred_mode_noextend_8x8[63:0] : vctrl_pred_mode_noextend_4x4[63:0]
vctrl_pred_mode_noextend_16x16[63:0]	vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] &

	vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0]
vctrl_pred_mode_noextend_8x8[63:0]	"h000" & vctrl_it_lumaintrapredmode0[15:12] & "h000" & vctrl_it_lumaintrapredmode0[11:8] & "h000" & vctrl_it_lumaintrapredmode0[7:4] & "h000" & vctrl_it_lumaintrapredmode0[3:0]
vctrl_pred_mode_noextend_4x4[63:0]	vctrl_it_lumaintrapredmode3[15:0] & vctrl_it_lumaintrapredmode2[15:0] & vctrl_it_lumaintrapredmode1[15:0] & vctrl_it_lumaintrapredmode0[15:0]

Inline data for RefPicSelect

	0	0	0	0 or 1	1	1	1
ExtendedForm	16x16	16x8	8x16	8x8	16x16	16x8	8x16
DW8 – 31:24	-	-	-	L0 blk3	L0 blk0	-	L0 blk1
DW8 – 23:16	-	-	-	L0 blk2	L0 blk0	-	L0 blk0
DW8 – 15:8	-	L0 blk1	L0 blk1	L0 blk1	L0 blk0	-	L0 blk1
DW8 – 7:0	L0 blk0	L0 blk0	L0 blk0	L0 blk0	L0 blk0	-	L0 blk0
DW9 – 31:24	-	-	-	L1 blk3	L1 blk0	-	L1 blk1
DW9 – 23:16	-	-	-	L1 blk2	L1 blk0	-	L1 blk0
DW9 – 15:8	-	L1 blk1	L1 blk1	L1 blk1	L1 blk0	-	L1 blk1
DW9 – 7:0	L1 blk0	L1 blk0	L1 blk0	L1 blk0	L1 blk0	-	L1 blk0

The inline data content of Dwords 4 to 6 is defined either for intra prediction or for inter prediction, but not both.

Inline data subfields for an Intra Macroblock

Dword	Bit	Description
7	31:16	**LumaIntraMode[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	**LumaIntraMode[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block (Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes.

Dword	Bit	Description
		4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes. See the bit assignment table later in this section.
8	31:16	**LumaIntraMode[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	**LumaIntraMode[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment later in this section.
9	31:8	Reserved: MBZ (Reserved for encocder turbo mode **IntraResidueDataSize**, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)
	7:0	**IntraStruct** This field contains 6 bits for IntraPredAvailFlags[5:0] and 2 bits for ChromaIntraPredMode. The IntraPredAvailFlags[4:0] (the lower 5 bits) have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbor position around the current MB or MB pair (in MBAFF mode). 1 – IntraPredAvailFlagY, indicates the values of samples of neighbor Y can be used in intra prediction for the current MB. 0 – IntraPredAvailFlagY, indicates the values of samples of neighbor Y is not available for intra prediction of the current MB. IntraPredAvailFlag-A and -E can only be different from each other when constrained_intra_pred_flag is equal to 1 and mb_field_decoding_flag is equal to 1 and the value of the mb_field_decoding_flag for the macroblock pair to the left of the current macroblock is equal to 0 (which can only occur when MbaffFrameFlag is equal to 1). IntraPredAvailFlag-F is used only if • It is in MBAFF mode, that is, **MbaffFrameFlag** = 1 • The current macroblock is of frame type, that is, **MbFieldFag** = 0 • The current macroblock type is Intra8x8, that is, **IntraMbFlag** = INTRA, **IntraMbMode** = INTRA_8x8, and **Transform8x8Flag** = 1 In any other cases IntraPredAvailFlag-A shall be used instead.

Bits	IntraPredAvailFlags Definition
7	**IntraPredAvailFlagF – F** (Left 8th row (-1,7) neighbor)

Dword	Bit	Description
		6 **IntraPredAvailFlagA – A** (Left neighbor top half) 5 **IntraPredAvailFlagE – E** (Left neighbor bottom half) 4 **IntraPredAvailFlagB – B** (Top neighbor) 3 **IntraPredAvailFlagC – C** (Top right neighbor) 2 **IntraPredAvailFlagD – D** (Top left corner neighbor) 1:0 **ChromaIntraPredMode –** 2 bits to specify 1 of 4 chroma intra prediction modes, see the table in later section.

Inline data subfields for an Inter Macroblock

DWord	Bit	Description
7	31:16	**Reserved: MBZ**
	15:8	**SubMbPredMode (Sub-Macroblock Prediction Mode):** If **InterMbMode** is INTER8x8, this field describes the prediction mode of the sub-partitions in the four 8x8 sub-macroblock. It contains four subfields each with 2-bits, corresponding to the four 8x8 sub-macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from **MbType** for a non-BP_8x8 inter macroblock, and carries redundant information as **MbType**). If **InterMbMode** is INTER16x16, INTER16x8 or INTER8x16, this field carries the prediction modes of the sub macroblock (one 16x16, two 16x8 or two 8x16). The unused bits are set to zero. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3]
	7:0	**SubMbShape** (Sub Macroblock Shape) This field describes the sub-block partitioning of each sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA). Otherwise, this field is ignored by hardware Bits [1:0]: SubMbShape[0] – for 8x8 Block 0

DWord	Bit	Description
		Bits [3:2]: SubMbShape[1] – for 8x8 Block 1 Bits [5:4]: SubMbShape[2] – for 8x8 Block 2 Bits [7:6]: SubMbShape[3] – for 8x8 Block 3 Blocks of the MB is numbered as follows : 01 23 Each 2-bit value [1:0] is defined as : 00 – SubMbPartWidth=8, SubMbPartHeight=8 01 – SubMbPartWidth=8, SubMbPartHeight=4 10 – SubMbPartWidth=4, SubMbPartHeight=8 11 – SubMbPartWidth=4, SubMbPartHeight=4
8	31:24	**RefPicSelect[0][3]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	23:16	**RefPicSelect[0][2]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	15:8	**RefPicSelect[0][1]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	7:0	**RefPicSelect[0][0]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
9	31:24	**RefPicSelect[1] [3]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.
	23:16	**RefPicSelect[1][2]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table.

DWord	Bit	Description
		For P- picture these bits must be set to zero.
	15:8	**RefPicSelect[1][1]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.
	7:0	**RefPicSelect[1][0]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.

Luma Intra Prediction Modes

Luma Intra Prediction Modes (LumaIntraPredModes) is defined in *Luma Intra Prediction Modes* . It is further categorized as Intra16x16PredMode (*Luma Intra Prediction Modes*), Intra8x8PredMode (*Luma Intra Prediction Modes*) and Intra4x4PredMode (*Luma Intra Prediction Modes*), operating on 16x16, 8x8 and 4x4 block sizes, respectively. illustrates the intra prediction directions geometrically for the Intra4x4 prediction. When a macroblock is subdivided, the intra prediction is performed for the subdivision in a predetermined order. For example, *Luma Intra Prediction Modes* shows the block order for Intra4x4 prediction. And *Luma Intra Prediction Modes* shows the block order of Block8x8 in a 16x16 region or Block4x4 in an 8x8 region.

Definition of LumaIntraPredModes

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	Bit	MbType = [1...24] Transform8x8Flag = 0	MbType = 0 Transform8x8Flag = 1	MbType = 0 Transform8x8Flag = 0
0	15:12	MBZ	**Block8x8 3**	**Block4x4 3 (0_0)**
	11:8	MBZ	**Block8x8 2**	**Block4x4 2 (0_1)**
	7:4	MBZ	**Block8x8 1**	**Block4x4 1 (0_2)**
	3:0	**Block16x16**	**Block8x8 0**	**Block4x4 0 (0_3)**
1	15:12	MBZ	MBZ	**Block4x4 7 (1_0)**
	11:8	MBZ	MBZ	**Block4x4 6 (1_1)**

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	Bit	MbType = [1...24] Transform8x8Flag = 0	MbType = 0 Transform8x8Flag = 1	MbType = 0 Transform8x8Flag = 0
	7:4	MBZ	MBZ	**Block4x4 5 (1_2)**
	3:0	MBZ	MBZ	**Block4x4 4 (1_3)**
2	15:12	MBZ	MBZ	**Block4x4 11 (2_0)**
	11:8	MBZ	MBZ	**Block4x4 10 (2_1)**
	7:4	MBZ	MBZ	**Block4x4 9 (2 2)**
	3:0	MBZ	MBZ	**Block4x4 8 (2_3)**
3	15:12	MBZ	MBZ	**Block4x4 15 (3_0)**
	11:8	MBZ	MBZ	**Block4x4 14 (3_1)**
	7:4	MBZ	MBZ	**Block4x4 13 (3_2)**
	3:0	MBZ	MBZ	**Block4x4 12 (3_3)**

Definition of Intra16x16PredMode

Intra16x16PredMode	Description
0	**Intra_16x16_Vertical**
1	**Intra_16x16_Horizontal**
2	**Intra_16x16_DC**
3	**Intra_16x16_Plane**
4 – 15	Reserved

Definition of Intra8x8PredMode

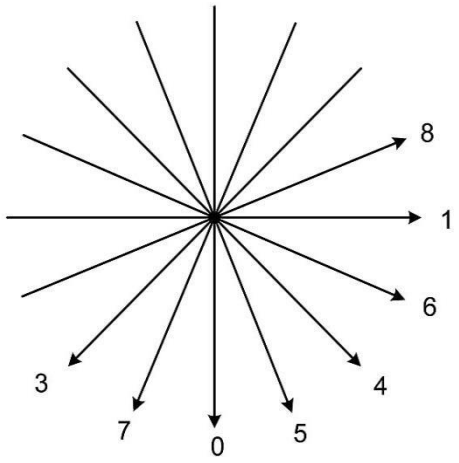
Intra8x8PredMode	Description
0	**Intra_8x8_Vertical**

Intra8x8PredMode	Description
1	**Intra_8x8_Horizontal**
2	**Intra_8x8_DC**
3	**Intra_8x8_Diagonal_Down_Left**
4	**Intra_8x8_Diagonal_Down_Right**
5	**Intra_8x8_Vertical_Right**
6	**Intra_8x8_Horizontal_Down**
7	**Intra_8x8_Vertical_Left**
8	**Intra_8x8_Horizontal_Up**
9 – 15	Reserved

Definition of Intra4x4PredMode

Intra4x4PredMode	Description
0	**Intra_4x4_Vertical**
1	**Intra_4x4_Horizontal**
2	**Intra_4x4_DC**
3	**Intra_4x4_Diagonal_Down_Left**
4	**Intra_4x4_Diagonal_Down_Right**
5	**Intra_4x4_Vertical_Right**
6	**Intra_4x4_Horizontal_Down**
7	**Intra_4x4_Vertical_Left**
8	**Intra_4x4_Horizontal_Up**
9 – 15	Reserved

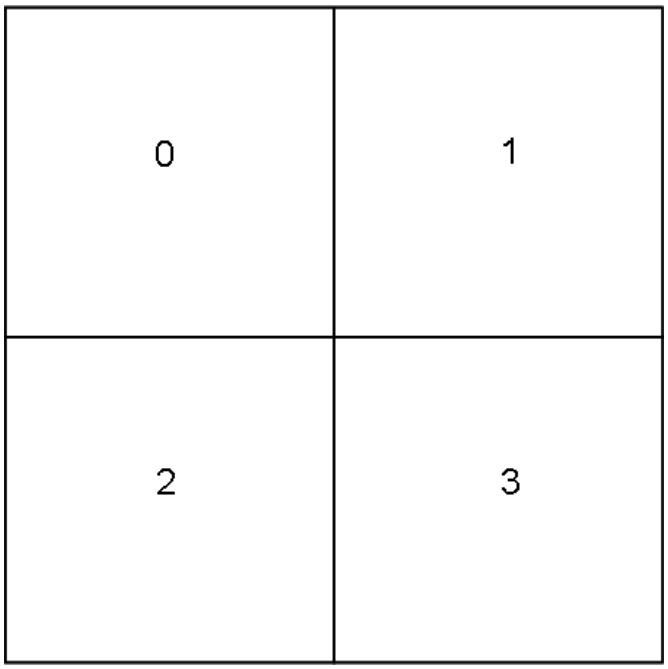
Intra_4x4 prediction mode directions



Numbers of Block4x4 in a 16x16 region

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region



Definition of Chroma Intra Prediction Mode

ChromaIntraPredMode (intra_chroma_pred_mode)	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Reference Indices Defined for Each MB Partition Type and Bit Assignment

	frame/field MB/Picture				
MB partitioning	16x16	16x8	8x16	8x8	
RefIdxL0/1[0]	blk0	blk0	blk0	blk0	Bit 7:0
RefIdxL0/1[1]	x	blk1	blk1	blk1	Bit 15:8
RefIdxL0/1[2]	x	x	x	blk2	Bit 23:16
RefIdxL0/1[3]	x	x	x	blk3	Bit 31:24

MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)

Current MB is labelled as X. For non-MBAFF mode, 4 neighbors, A, B, C, D, are depicted in the following picture and are defined as the following.

- MB D: top left neighbor of current MB X

- MB C: top right neighbor of current MB X
- MB B: top neighbor of current MB X
- MB A: left neighbor of the current MB X

mbAddrD D (top-left)	**mbAddrB** B (top)	**mbAddrC** C (top-right)
mbAddrA A (left)	**CurrMbAddrX** X	N/A
N/A	N/A	N/A

For MBAFF mode, the current MB is labelled as X0 or X1, 4 neighbor pairs, A0/A1, B0/B1, C0/C1, D0/D1, are depicted in the following picture and are defined as the following.

- MB D0: first MB of top left neighbor MB pair of current MB pair X0/X1
- MB D1: second MB of top left neighbor MB pair of current MB pair X0/X1
- MB C0: first MB of top right neighbor MB pair of current MB pair X0/X1
- MB C1: second MB of top right neighbor MB pair of current MB pair X0/X1
- MB B0: first MB of top neighbor MB pair of current MB pari X0/X1
- MB B1: second MB of top neighbor MB pair of current MB pari X0/X1
- MB A0: first MB of left neighbor MB pair of the current MB pair X0/X1
- MB A1: second MB of left neighbor MB pair of the current MB pair X0/X1

mbAddrD D0	**mbAddrB** B0	**mbAddrC** C0
mbAddrD+1 D1	**mbAddrB+1** B1	**mbAddrC+1** C1
mbAddrA A0	**CurrMbAddrX** X0 or	N/A
mbAddrA+1 A1	**CurrMbAddrX** X1	N/A

For a given macroblock X (or X0/X1), the 6 neighbor availability signals, namely, A, B, C, D, E, F, are defined as the following.

- IntraPredAvailFlagF – F (Single neighbor pixel at the left 8th row (-1,7)
- IntraPredAvailFlagA – A (Left neighbor top half pixel group)
- IntraPredAvailFlagE – E (Left neighbor bottom half pixel group)
- IntraPredAvailFlagB – B (Top neighbor pixel group)
- IntraPredAvailFlagC – C (Top right neighbor pixel group)
- IntraPredAvailFlagD – D (Top left corner neighbor pixel)

The following table depicts the generation of IntraPredAvailFlags[5:0] signals in a condensed form. It should note that for most cases only one input neighbor signal is assigned for each condition. The exception is in the four places for deriving left neighbor A and E where the neighbor is only available if left neighbors (A0 and A1) are both available (A0&A1). Also note that F takes output value very similar to that for A except the two "AND" conditions, where F is assigned to A1 instead of (A0&A1).

Table: Definition of intra-prediction neighbor availability calculation in MBAFF mode

Output ⇒		**D**		**B**		**C**		**A**		**E**		**F**	
Current X \ Neighbor Y		**Y-Frame**	**Y-Field**	**Y-Frame**	**Y-Field**	**Y-Frame**	**Y-Field**	**Y-Frame**	**Y-Field**	**Y-Frame**	**Y-Field**	**Y-Frame**	**Y-Field**
X_0 (Top)	X-Frame	**D_1**	**D_1**	**B_1**	**B_1**	**C_1**	**C_1**	**A_0**	**A_0 & A_1**	**A_0**	**A_0 & A_1**	**A_0**	**A_1**
	X-Field	**D_1**	**D_0**	**B_1**	**B_0**	**C_1**	**C_0**	**A_0**	**A_0**	**A_1**	**A_0**	**A_0**	**A_0**
X_1 (Bottom)	X-Frame	**A_0**	**A_1**	**X_0**	**N/A**	**0**	**0**	**A_1**	**A_0 & A_1**	**A_1**	**A_0 & A_1**	**A_1**	**A_1**

Output ⇒		D		B		C		A		E		F	
Current X \ Neighbor Y		Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field
	X-Field	D_1	D_1	B_1	B_1	C_1	C_1	A_0	A_1	A_1	A_1	A_0	A_1

In *MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)* , *X-Frame* or *X-Field* indicates the frame/field mode of the current MB; and *Y-Frame* or *Y-Field* indicates the corresponding neighbor MB for the given neighbor location, being upper left (D) or left (A) for example. Therefore, "Y-" takes the selected neighbor MB name as in the output cell in the same column. For example, for output D, if X1 is a frame MB, Y = A, if X1 is a field MB, Y = D.

For non-MBAFF mode, as A0=A1, B0=B1, C0=C1 and D0=D1, the neighbor assignment is degenerated into the following simple table. Here, E is assigned to the same as A and F is forced to 0.

Table: Definition of intra-prediction neighbor availability calculation in non-MBAFF mode

Output ⇒	D	B	C	A	E	F
X	D0	B0	C0	A0	A0	0

To further explain the neighbor assignment rules in *MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)* , the following table provides description for each condition. Please note that this table is **informative** as it provides redundant information as in *MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)* .

Table: Detailed explanation of intra-prediction neighbor availability calculation in MBAFF mode

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
X0 (Top)	X-Frame	Y-Frame	D	D1	Top Frame MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
	X-Frame	Y-Field	D	D1	
	X-Field	Y-Frame	D	D1	Top Field MB uses [-1,-2] = D_30, thus if D is frame pair, takes D1 (D1_14 pixel), and if D is field pair, takes D0 (D0_15 pixel)
	X-Field	Y-Field	D	D0	
X1 (Bottom)	X-Frame	Y-Frame	A	A0	Bottom Frame MB uses [-1,15] = A_15, thus A0 (A0_15 pixel) if A is a frame pair, or A1 (A1_7 pixel), if A is a field pair
	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	D	D1	Bottom Field MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
	X-Field	Y-Field	D	D1	
			B		
X0 (Top)	X-Frame	Y-Frame	B	B1	Top Frame MB uses [0...15,-1] = B_31, thus B1 only, regardless B frame or field pair
	X-Frame	Y-Field	B	B1	
	X-Field	Y-Frame	B	B1	Top Field MB uses [0...15,-2] = B_30, thus if B

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
	X-Field	Y-Field	B	B0	is frame pair, takes B1 (B1_14 row), and if B is field pair, takes B0 (B0_15 row)
X1 (Bottom)	X-Frame	Y-Frame	X	X0	Bottom Frame MB uses [0…15,15], thus X0 (X0_15 row)
	X-Frame	Y-Field	X	n/a	**Note:** X0 and X1 must have the same field type, this row is n/a.
	X-Field	Y-Frame	B	B1	Bottom Field MB uses [0…15,-1] = B_31, thus B1 only, regardless B frame or field pair
	X-Field	Y-Field	B	B1	
			C		
X0 (Top)	X-Frame	Y-Frame	C	C1	Top Frame MB uses [16…23,-1] = C_31, thus C1 only, regardless C frame or field pair
	X-Frame	Y-Field	C	C1	
	X-Field	Y-Frame	C	C1	Top Field MB uses [16…23,-2] = C_30, thus if C is frame pair, takes C1 (C1_14 row), and if C is field pair, takes C0 (C0_15 row)
	X-Field	Y-Field	C	C0	
X1 (Bottom)	X-Frame	Y-Frame	n/a	0	Bottom Frame MB doesn't have left-top neighbor by definition, thus forced to 0
	X-Frame	Y-Field	n/a	0	
	X-Field	Y-Frame	C	C1	Bottom Field MB uses [16…23,-1] = C_31, thus C1 only, regardless C frame or field pair
	X-Field	Y-Field	C	C1	
			A		
X0 (Top)	X-Frame	Y-Frame	A	A0	First Half of Top Frame MB uses [-1,0…7], thus A0 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Top Field MB uses [-1,0..2..4..14], thus take A0 (if A is frame pair, takes A0 even lines, and if A is field pair, takes A0 first half)
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	First Half of Bottom Frame MB uses [-1,16…23], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Bottom Field MB uses [-1,1..3..15], thus take A0 (if A is frame pair, takes A0 odd lines, and if A is field pair, takes A1 first half)
	X-Field	Y-Field	A	A1	
			E		
X0 (Top)	X-Frame	Y-Frame	A	A0	Second Half of Top Frame MB uses [-1,8…15], thus A0 if A is a frame pair; but is only avail if
	X-Frame	Y-Field	A	A0&A1	

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
					both A0 and A1 are avail if A is a field pair due to the mix
	X-Field	Y-Frame	A	A1	Second Half of Top Field MB uses [-1,16..18..30], thus take A1 (if A is frame pair, takes A1 even lines, and if A is field pair, takes A0 second half)
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	Second Half of Bottom Frame MB uses [-1,24...31], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A1	Second Half of Bottom Field MB uses [-1,17..19..31], thus takes A1 (if A is frame pair, takes A1 odd lines, and if A is field pair, takes A1 second half)
	X-Field	Y-Field	A	A1	
			F		
X0 (Top)	X-Frame	Y-Frame	A	A0	Top Frame MB uses [-1,7] = A_7 (odd location), thus A0 if A is frame pair and A1 if field pair
	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Top Field MB uses [-1,14] = A_14 (even location), thus A0 regardless A frame or field pair
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	Bottom Frame MB uses [-1,23] = A_23 (odd location), thus A1 regardless A frame or field pair
	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Bottom Field MB uses [-1,15] = A_15 (odd location), thus A0 if A is frame pair and A1 if A is field pair
	X-Field	Y-Field	A	A1	

Macroblock Type for Intra Cases

MbType follows two different tables according to whether the macroblock is an inter or intra macroblock according to IntraMbFlag.

For an intra macroblock, MbType, as defined in *Macroblock Type for Intra Cases* , carries redundant information as IntraMbMode. The notation I_16x16_x_y_z used in the table, 'x' is Intra16x16LumaPredMode, 'y' is ChromaCbpInd, and 'z' is LumaCbpInd, as defined in *Macroblock Type for Intra Cases* .

MbType definition for Intra Macroblock

Macroblock Type	MbType
I_4x4	0
I_8x8	0
I_16x16_0_0_0	1
I_16x16_1_0_0	2
I_16x16_2_0_0	3
I_16x16_3_0_0	4
I_16x16_0_1_0	5
I_16x16_1_1_0	6
I_16x16_2_1_0	7
I_16x16_3_1_0	8
I_16x16_0_2_0	9
I_16x16_1_2_0	Ah
I_16x16_2_2_0	Bh
I_16x16_3_2_0	Ch
I_16x16_0_0_1	Dh
I_16x16_1_0_1	Eh
I_16x16_2_0_1	Fh
I_16x16_3_0_1	10h
I_16x16_0_1_1	11h
I_16x16_1_1_1	12h
I_16x16_2_1_1	13h
I_16x16_3_1_1	14h

Macroblock Type	MbType
I_16x16_0_2_1	15h
I_16x16_1_2_1	16h
I_16x16_2_2_1	17h
I_16x16_3_2_1	18h
I_PCM	19h (used by HW)

Note: MbType here is identical as specified in DXVA 2.0.

For Intra_16x16 modes, the 5 bits of value (MbType – 1) have the following meanings.

Sub field definition used by MbType for a macroblock with Intra16x16 prediction

Bits	Description
4	**LumaCbpInd** – Luma Coded Block Pattern Indicator 0 means none of the luma blocks are coded. 1 means that at least one luma block is coded. 0 = SUBMODE_I16_L_0 1 = SUBMODE_I16_L_NZ In VME output, this field is forced to be 1 before adding 1 in Intra_16x16 mode.
3:2	**ChromaCbpInd** – Chroma Coded Block Pattern Indicator 00 means none of chroma blocks are coded. 01 means that only the chroma DC block is coded, but all AC blocks are not coded. 10 means that at least one AC chroma block is coded. 00 = SUBMODE_I16_C_0 01 = SUBMODE_I16_C_DC 10 = SUBMODE_I16_C_NZ 11 = Reserved In VME output, this field is forced to be 10 before adding 1 in Intra_16x16 mode. *Programming Note: Adding 1 to MbType by VME hardware may have carry in to this field. But as '11' is reserved, the carry-in doesn't propagate into bit 4 or higher. This allows software to update MbType, if desired, using the redundant LumaIntraPredModes information.*
1:0	**Intra16x16PredMode** – Intra16x16 Prediction Mode These two bits carries redundant (identical) information as that in

Bits	Description
	LumaIntraPredModes[0][0]. 0 = SUBMODE_I16_VER 1 = SUBMODE_I16_HOR 2 = SUBMODE_I16_DC 3 = SUBMODE_I16_PLANE

IntraMbMode definition

IntraMbMode [1:0]	Description	Supported by VME?	Used by PAK?
0	**INTRA_16x16 (redundant with MbType)**	Yes	Ignored
1	**INTRA_8x8**	Yes	Yes
2	**INTRA_4x4**	Yes	Yes
3	**IPCM (redundant with MbType)**	No	Ignored

As an alternative representation, MbType is logically the same as the following, except the I_PCM and I_NxN (i.e. I_4x4 and I_8x8) cases:

- 24 types of 16x16 intra modes: **A**+**B**+**C**+**D**: (1h – 18h)

  MBTYPE_INTRA_16x16 1h A

  - 4 Intra16x16 modes:

    SUBMODE_I16_VER 0 B
    SUBMODE_I16_HOR 1 B
    SUBMODE_I16_DC 2 B
    SUBMODE_I16_PLN 3 B

  - 3 Chroma Cbp indices:

    SUBMODE_I16_C_0 0 C
    SUBMODE_I16_C_DC 4 C
    SUBMODE_I16_C_NZ 8 C

  - 2 Luma Cbp indices:

    SUBMODE_I16_L_0 0 D
    SUBMODE_I16_L_NZ Ch D

Macroblock Type for Inter Cases

Sub-Macroblock Prediction Mode, SubMbPredMode, indicates the prediction mode for the sub-partitions. Prediction mode specifies prediction direction being forward (from L0), backward (from L1) or bi-directional (from both L0 and L1). Its meaning depends on InterMbMode. *Macroblock Type for Inter Cases* provides the definition of the field.

- If InterMbMode is INTER16x16, only SubMbPredMode[0] is valid, it describes the prediction mode of the 16x16 macroblock. The other entries are set to zero by hardware.
    - For AVC, SubMbPredMode[0] contains redundant information as encoded in MbType parameter.
    - *Note: SubMbPredMode[1]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.*
- If InterMbMode is INTER16x8, and INTER8x16, only the first two entries SubMbPredMode[0] and SubMbPredMode[1] are valid, describing the sub-macroblock prediction mode.
    - For AVC, SubMbPredMode[0]/[1] contains redundant information as encoded in MbType parameter.
    - *Note: SubMbPredMode[2]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.*
- If InterMbMode is INTER8x8, each entry of SubMbPredMode describes the prediction mode of the sub-partition of an 8x8 sub-macroblock.
    - For AVC, SubMbPredMode can be derived from sub_mb_type field for BP_8x8 macroblocks as defined in AVC spec.
    - *Note on Direct Sub-macroblock Prediction Mode: Direct prediction is not conveyed through SubMbPredMode, instead, it is carried through Direct8x8Pattern.*

InterMbMode definition

MbSkipFlag	InterMbMode	Description
0	0	**INTER16x16**
0	1	**INTER16x8**
0	2	**INTER8x16**
0	3	**INTER8x8**
1	0	**PSKIP/BSKIP16x16***
1	3	**BSKIP**
1	1, 2	Reserved
Used by PAK	Ignored by PAK	

* BSKIP16x16 is an optional non-standard but equivalent optimization.

Definition of SubMbPredMode based on InterMbMode

SubMbPredMode	INTER16x16	INTER16x8	INTER8x16	INTER8x8
Bit	MbType = [1...3]	MbType = [16h]	MbType = [4...15h]	MbType = [16h]
7:6	MBZ	MBZ	MBZ	**Block8x8 3**
5:4	MBZ	MBZ	MBZ	**Block8x8 2**
3:2	MBZ	**Block16x8 1**	**Block8x16 1**	**Block8x8 1**
1:0	**Block16x16**	**Block16x8 0**	**Block8x16 0**	**Block8x8 0**
	Ignored by PAK	**Ignored by PAK**	**Ignored by PAK**	**Used by PAK**

Definition of SubMbPredMode[i]

SubMbPredMode	Description	InterMbMode	VME Output	MvCountPred	Notes
0	Pred_L0	All	Yes	1	P or B Slice
1	Pred_L1	All	Yes	1	B Slice Only
2	BiPred	All	Yes	2	B Slice Only
3	Reserved	Reserved	Reserved	Reserved	Reserved

Sub-Macroblock Shape, SubMbShape[i], for i = 0...3, describes the shape of the sub partitions of the 8x8 sub-macroblock of a BP_8x8 macroblock. This field is only valid if InterMBMode is INTER8x8. They are defined in *Macroblock Type for Inter Cases* . The parameters can be derived from *sub_mb_type* field as defined in AVC spec.

Note: These fields must be correctly set even for **Direct** or **Skip** 8x8 cases, the individual B_Direct_8x8 block is flagged by the **Direct8x8Pattern** variable.

Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock (including BSKIP, BDIRECT)

	Description			
SubMbShape	NumSubMbPart	SubMbPartWidth	SubMbPartHeight	MvCountShape
0	1	8	8	1
1	2	8	4	2
2	2	4	8	2
3	4	4	4	4

For an inter macroblock, MbType, carries redundant information as InterMbMode and SubMbPredMode. *Macroblock Type for Inter Cases* provides the typical inter macroblock types and *Macroblock Type for Inter Cases* provides that with skip and direct modes. The definition of MbType for both P slice and B

slice is the same and is equivalent to that for mb_type of a B slice in the AVC spec. As direct mode is indicated using a separate field Direct8x8Pattern, 0 is reserved for MbType.

Here, MVCount is the number of motion vectors actually encoded in the bitstream. It is informative. For a BP_8x8 or equivalent Skip/Direct macroblock, MVCount is the sum of the following term for the four 8x8 sub macroblock (with i = 0...3):

MvCountShape[i] * MvCountPred[i] * MvCountDirect[i]

where MvCountShape[i] is block count for sub macroblock [i], MvCountPred[i] is the motion vector count for each block of sub macroblock[i], and MvCountDirect[i] is the multipler for direct mode for B Slice, indicating whether motion vectors are coded or not. It must be set to 1 for P slice. For B Slice, MvCountDirect[i] = !Direct8x8Pattern[i], which is 0 for a sub macroblock coded as direct mode and 1 otherwise.

In the tables, "DC" stands for "Don't Care" as PAK hardware ignores these fields.

MbType definition for Inter Macroblock (and MbSkipflag = 0)

Macroblock Type	MbType	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MVCount
Reserved	0	-	-	-	-	-
BP_L0_16x16	1	0	0	DC	DC	1
B_L1_16x16	2	0	0	DC	DC	1
B_Bi_16x16	3	0	0	DC	DC	2
BP_L0_L0_16x8	4	0	0	DC	DC	2
BP_L0_L0_8x16	5	0	0	DC	DC	2
B_L1_L1_16x8	6	0	0	DC	DC	2
B_L1_L1_8x16	7	0	0	DC	DC	2
B_L0_L1_16x8	8	0	0	DC	DC	2
B_L0_L1_8x16	9	0	0	DC	DC	2
B_L1_L0_16x8	0Ah	0	0	DC	DC	2
B_L1_L0_8x16	0Bh	0	0	DC	DC	2
B_L0_Bi_16x8	0Ch	0	0	DC	DC	3
B_L0_Bi_8x16	0Dh	0	0	DC	DC	3
B_L1_Bi_16x8	0Eh	0	0	DC	DC	3
B_L1_Bi_8x16	0Fh	0	0	DC	DC	3
B_Bi_L0_16x8	10h	0	0	DC	DC	3
B_Bi_L0_8x16	11h	0	0	DC	DC	3
B_Bi_L1_16x8	12h	0	0	DC	DC	3
B_Bi_L1_8x16	13h	0	0	DC	DC	3
B_Bi_Bi_16x8	14h	0	0	DC	DC	4
B_Bi_Bi_8x16	15h	0	0	DC	DC	4

Macroblock Type	MbType	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MVCount
BP_8x8	16h	0	!= Fh	vary	vary	Sum
Reserved	17h-1Fh	-	-	-	-	-

Additional MbType definition with Direct/Skip for Inter Macroblock

Macroblock Type	Mb Type	Xfrm 8x8	MbSkip Flag	Direct8x8 Pattern	SubMb Shape	SubMb PredMode	MvCount	Notes
P_Skip_16x16	1	-	1	DC	DC	DC	0	Skipped macroblock. Motion compensation like P_L0_16x16
B_Skip_16x16_4MVPair	16h	vary	1	Fh	0	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 8x8 subblocks, when **direct_8x8_inference_flag** is set to 1
B_Skip_16x16_16MVPair	16h	0	1	Fh	FFh	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 4x4 subblocks, when **direct_8x8_inference_flag** is set to 0
B_Direct_16x16_4MVPair	16h	vary	0	Fh	0	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 8x8 subblocks, when **direct_8x8_inference_flag** is set to 1
B_Direct_16x16_16MVPair	16h	0	0	Fh	FFh	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 4x4 subblocks, when **direct_8x8_inference_flag** is set to 0

People might notice that B_DIRECT_16x16 and B_SKIP are mapped on BP_8x8 for AVC decoding interface in IT mode as the motion compensation operation for both modes are the same as BP_8x8. According to AVC Spec, motion vectors for B_DIRECT_16x16 and B_SKIP are derived from temporally co-located macroblock on an 8x8 sub macroblock basis if direct_8x8_inference_flag is set to 1 or on a 4x4 block basis if it is set to 0. For each sub macroblock or block, SubMbPredMode is derived, thus can any of the valid numbers. Motion vectors may also be different. In spatial direct mode, the motion vectors are subject to spatial neighbor macroblocks as well as co-located macroblock. The spatial prediction is based on the neighbor macroblocks, so the same spatial predicted motion vector applies to all sub macroblocks or blocks. However, under certain conditions, temporal predictor may replace (colZeroFlag) the spatial predictor for a given sub macroblock or block. Thus the motion vectors may differ.

In *Macroblock Type for Inter Cases* , the macroblock type names for major partitions nicely follow forms *BP_MbPredMode_MbShape* (like BP_L0_16x16) and *B_MbPredMode0_MbPredMode1_MbShape* (like B_L0_Bi_16x8). For minor partitions it is fixed as *BP_MbShape* as BP_8x8.

However, in *Macroblock Type for Inter Cases* the macroblock types for Skip and Direct modes does not follow the same rule. The third field in P_Skip_16x16 or B_Direct_16x16_x indicates that "Skip" or "Direct" applies to the entire 16x16 macroblock, even though MbShape is 8x8 as that in BP_8x8. In order to distinguish the SubMbShape being 8x8 or 4x4 for B_Skip and B_Direct, the fourth field is added. 4MVP**air** indicates upto 4 MV pairs are presented with SubMbShape equals to 0; and 16MVP**air** indicates up to 16 MV pairs are presented with SubMbShape equals to FFh.Also note that P_8x8ref0 is not specified in PAK input interface, it is up to hardware to detect and choose its packing format based on number of reference indices and reference index for the given macroblock.

Macroblock Type Conversion Rules

For improved coding efficiency the PAK hardware has the capability to convert macroblock types to use more efficiency coding modes such as DIRECT and SKIP. For an inter macroblock or a sub macroblock coded as DIRECT, no motion vector is needed in the bitstream for the macroblock or sub macroblock. If a macroblock is coded as SKIP, it only consumes one SKIP bit (no motion vector, no coefficients are coded). And infomaton about the macroblock is 'inferred' according to the rules stated in the AVC Spec.

As the input to PAK, the following signals can convey the information regarding DIRECT and SKIP:

- MbSkipFlag
- Direct8x8Pattern
- CodecBlockPattern (CbpY, CbpCb, CbpCr)

Such conversion can be enabled or disabled through the SLICE_STATE fields DirectConvDisable and SkipConvDisable as well as the in line command field MbSkipConvDisable.

A P slice doesn't support direct mode, it only supports P_Skip, which is equivalent to a 16_16_L0 prediction. Other prediction types cannot be converted to P_Skip. The following table describes the macroblock type conversion rules for a P slice. Here CBP = CbpY/CbpCb/CbpCr are the final computed results after quantization by the hardware. Note that hardware honors the input CbpY/CbpCb/CbpCr fields - if the value corresponding to a block is set to zero, the resulting CBP is also zero. The output

mb_skip_flag and mb_type are the symbols coded in the bitstream as defined in the AVC spec. *DC* stands for *Don't care*, *T* for *True*.

Note that the internal condition of MV==MVP is subject to the precise rules stated in the AVC Spec as quoted below. Note that there are exceptions for P_Skip from the normal motion vector prediction rules.

Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0. (8-168)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

- The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

- The variable mvL0 is specified as follows.

- If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

- mbAddrA is not available

- mbAddrB is not available

- refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

- refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

- Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE - The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

Macroblock type conversion rule for an inter macroblock in a P slice

Input		Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
P_Skip_16x16	DC	DC	DC	1	1	-	Forced to P_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control. Hardware doesn't check for

Input		Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
							MV==MVP error condition
P_Skip_16x16	DC	DC	DC	0	0	0	Reverse convert to P_L0_16x16; Hardware will force CBP to zero but reversely convert MbType as P_L0_16x16 once it determines that Skip is not allowed.
BP _16x16_L0	0	0	T	1	1	-	Converted to P_Skip. Even input doesn't provide skip hint, hardware can performance the optimization by detecting CBP and MV==MVP condition.
BP _16x16_L0	0	0	T	0	0	0	Reverse back to P_L0_16x16; Hardware will reverse back to P_L0_16x16 even Skip conditions are met once it determines that Skip is not allowed.
BP _16x16_L0	1	0	T	T	0	0	Still coded as P_L0_16x16 = 0.

A B slice supports both direct and skip modes. The following table describes the macroblock type conversion rules for a B slice. Hardware does not verify MV==MVP condition for a Skip/Direct macroblock in a B Slice as no DMV is performed by hardware.

Macroblock type conversion rule for an inter macroblock in a B slice

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	1	1	-	Forced to B_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control.
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	0	0	0	REVERSE convert to B_Direct_16x16; Hardware will force CBP to zero and also reverse convert to B_Direct_16x16 when it discovers Skip is not allowed.
B_Direct_16x16_4MVPair/ 16MVPair	0	0	0	n/a	1	1	-	Converted to B_Skip. Hardware first converts to B_Direct_16x16 and then further to B_Skip if CBP = 0.
B_Direct_16x16_4MVPair/ 16MVPair	0	0	0	n/a	0	0	0	Converted to B_Direct_16x16. Hardware

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
								first converts to B_Direct_16x16 and stop there as it discovers Skip is not allowed even CBP=0.
B_Direct_16x16_4MVPair/ 16MVPair	1	0	0	n/a	DC	0	0	Converted to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there even though CBP = 0 as input disallows Skip conversion.
B_Direct_16x16_4MVPair/ 16MVPair	DC	0	NZ	n/a	DC	0	0	Converted to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there because CBP != 0.
B_Direct_16x16_4MVPair/ 16MVPair	DC	1	DC	n/a	DC	0	16h	Stay as B_8x8. Hardware

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
								stays at B_8x8 and codes each sub macroblocks even all are direct.

The internal signal MbAffSkipAllowed is added to deal with a restriction on the frame/field flag (MbFieldFlag) which is unique to MBAFF. MbAffSkipAllowed is always set to 1 in non-MBAFF modes. In MBAFF mode, a macroblock pair may be both skipped only if its MbFieldFlag is the same as its available neighbor macroblock pair A or B if A or B is available (in that order), or is not 0 if A/B are both not available. Otherwise, one of the macroblocks in the pair must be coded.

To reduce the burden on software, PAK hardware handles the above restriction correctly. For the first MB in a pair, MbAffSkipAllowed is always set to 1. Therefore, hardware allows converting the first MB to Skip if skip conversion is enabled. For the second MB in a pair, hardware sets MbAffSkipAllowed to 0 if the following is true:

- The current MB Pair has different MbFieldFlag than its available neighbor A or B if A or B is available, or is not 0 if A/B are both not available
- And the first MB is coded as a SKIP (could be forced or converted)

Otherwise, it sets MbAffSkipAllowed to 1. As MbAffSkipAllowed is to 0 for the above condition, hardware will disallow Skip mode for the second MB. If the input signal forces it to Skip, hardware performs reverse-convertion to code it as P_L0_16x16 or B_Direct_16x16 with CBP = 0 for a macroblock in a P or B Slice. This means that hardware is able to correct the programming mistake by software. If the macroblock is not forced to skip, hardware simply disallows Skip conversion.

Software still has an option to disallow Skip Conversion on a per-MB basis using the MbSkipConvDisable control field in the inline command.

Indirect Data Description

For each macroblock, an ENC-PAK data set consists of two types of data blocks: indirect **MV data block** and **inline MB information**.

The indirect MV data block may be in two modes: **unpackedmode** and **packed-size mode**.

Unpacked Motion Vector Data Block

In the **unpacked** mode, motion vectors are expanded (or duplicated) to either bidirectional 8x8 8MV major partition format, or bidirectional 4x4 32MV format. Thus either 32 bytes or 128 bytes is assigned to each MB.

Motion Vector block contains motion vectors in an intermediate format that is partially expanded according to the sub- macroblock size. During the expansion, a place that does not contain a motion vector is filled by replicating the relevant motion vector according to the following motion vector replication rules. If the relevant motion vector doesn't exist (for the given L0 or L1), it is zero filled.

Motion Vector Replication Rules:

- Rule #1
    - #1.1: For L0 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L0 inter prediction exists, the corresponding L0 MV is used
        - Else it must be zero
    - #1.2: For L1 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L1 inter prediction exists, the corresponding L1 MV is used
        - Else it must be zero
- For a macroblock with a 16x16, 16x8 or 8x16 sub-macroblock, MvSize = 8. The eight MV fields follow Rule #1.
    - The 16x16 is broken down into 4 8x8 sub-macroblocks. The 16x16 MVs (after rule #1) are replicated into all 8x8 blocks.
    - For an 8x16 partition, each 8x16 is broken down into 2 8x8 stacking vertically. The 8x16 MVs (after rule #1) are replicated into both 8x8 blocks.
    - For a 16x8 partition, each 16x8 is broken down into 2 8x8 stacking horizontally. The 16x8 MVs (after rule #1) are replicated into both 8x8 blocks.
- For macroblock with sub-macroblock of 8x8 without minor partition (SubMbShape[0…3] = 0), MvSize = 8, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
    - There is no motion vector replication
- For macroblock with sub-macroblock of 8x8 with at least one minor partition (if any SubMbShape[i] != 0), MvSize = 32, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
    - For an 8x8 sub-partition, the 8x8 MVs (after rule #1) is replicated into all the four 4x4 blocks.
    - For an 4x8 sub-partition within an 8x8 partition, each 4x8 is broken down into 2 4x4 stacking vertically. The 4x8 MVs (after rule #1) are replicated into both 4x4 blocks.

- For an 8x4 sub-partition within an 8x8 partition, each 8x4 is broken down into 2 4x4 stacking horizontally. The 8x4 MVs (after rule #1) are replicated into both 4x4 blocks.
- For a 4x4 sub-partition within an 8x8 partition, each 4x4 has its own MVs (after rule #1).

Motion Vector block and MvSize

DWord	Bit	MvSize	
		8	32
W1.0	31:16	**MV_Y0_L0.y**	**MV_Y0_0_L0.y**
	15:0	**MV_Y0_L0.x**	**MV_Y0_0_L0.x**
W1.1	31:16	**MV_Y0_L1.y**	**MV_Y0_0_L1.y**
	15:0	**MV_Y0_L1.x**	**MV_Y0_0_L1.x**
W1.2	31:0	**MV_Y1_L0**	**MV_Y0_1_L0**
W1.3	31:0	**MV_Y1_L1**	**MV_Y0_1_L1**
W1.4	31:0	**MV_Y2_L0**	**MV_Y0_2_L1**
W1.5	31:0	**MV_Y2_L1**	**MV_Y0_2_L0**
W1.6	31:0	**MV_Y3_L0**	**MV_Y0_3_L0**
W1.7	31:0	**MV_Y3_L1**	**MV_Y0_3_L1**
W2.0	31:0	**n/a**	**MV_Y1_0_L1**
W2.1	31:0	**n/a**	**MV_Y1_0_L0**
W2.2	31:0	**n/a**	**MV_Y1_1_L1**
W2.3	31:0	**n/a**	**MV_Y1_1_L0**
W2.4	31:0	**n/a**	**MV_Y1_2_L1**
W2.5	31:0	**n/a**	**MV_Y1_2_L0**
W2.6	31:0	**n/a**	**MV_Y1_3_L0**
W2.7	31:0	**n/a**	**MV_Y1_3_L1**

DWord	Bit	MvSize	
		8	32
W3.0	31:0	**n/a**	**MV_Y2_0_L1**
W3.1	31:0	**n/a**	**MV_Y2_0_L0**
W3.2	31:0	**n/a**	**MV_Y2_1_L1**
W3.3	31:0	**n/a**	**MV_Y2_1_L0**
W3.4	31:0	**n/a**	**MV_Y2_2_L1**
W3.5	31:0	**n/a**	**MV_Y2_2_L0**
W3.6	31:0	**n/a**	**MV_Y2_3_L0**
W3.7	31:0	**n/a**	**MV_Y2_3_L1**
W4.0	31:0	**n/a**	**MV_Y3_0_L1**
W4.1	31:0	**n/a**	**MV_Y3_0_L0**
W4.2	31:0	**n/a**	**MV_Y3_1_L1**
W4.3	31:0	**n/a**	**MV_Y3_1_L0**
W4.4	31:0	**n/a**	**MV_Y3_2_L1**
W4.5	31:0	**n/a**	**MV_Y3_2_L0**
W4.6	31:0	**n/a**	**MV_Y3_3_L0**
W4.7	31:0	**n/a**	**MV_Y3_3_L1**

The motion vector(s) for a given sub-macroblock or a sub-partition are uniquely placed in the output message as shown by the non-duplicate fields in *Unpacked Motion Vector Data Block* and *Unpacked Motion Vector Data Block*.

MV_Yx_L0 and MV_Yx_L1 may be present individually or both. If one is not present, the corresponding field must be zero. Subsequently, the duplicated fields will be zero as well.

Motion Vector duplication by sub-macroblocks for a 16x16 macroblock, whereas the 8x8 column is for 4x(8x8) partition without minor shape

DWord	Bit	16x16	16x8	8x16	8x8
W1.0	31:16	**MV_Y0_L1 (A)**	**MV_Y0_L1 (A)**	**MV_Y0_L1**	**MV_Y0_L1**
	15:0	**MV_Y0_L0 (A)**	**MV_Y0_L0 (A)**	**MV_Y0_L0**	**MV_Y0_L0**
W1.1	31:16	**Duplicate (A)**	**Duplicate (A)**	**MV_Y1_L1**	**MV_Y1_L1**
	15:0	**Duplicate (A)**	**Duplicate (A)**	**MV_Y1_L0**	**MV_Y1_L0**
W1.2	31:16	**Duplicate (A)**	**MV_Y2_L1 (B)**	**Duplicate (A)**	**MV_Y2_L1**
	15:0	**Duplicate (A)**	**MV_Y2_L0 (B)**	**Duplicate (A)**	**MV_Y2_L0**
W1.3	31:16	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y3_L1**
	15:0	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y3_L0**

Motion Vector duplication by sub-partitions for the first 8x8 sub-macroblock Y0 if any Y0-Y3 contains minor shape (Y1_ to Y3_ have the same format in W2 to W4)

DWord	Bit	8x8	8x4	4x8	4x4
W1.0	31:16	**MV_Y0_L1**	**MV_Y0_0_L1 (A)**	**MV_Y0_0_L1 (A)**	**MV_Y0_0_L1**
	15:0	**MV_Y0_L0**	**MV_Y0_0_L0 (A)**	**MV_Y0_0_L0 (A)**	**MV_Y0_0_L0**
W1.1	31:16	**Duplicate (A)**	**Duplicate (A)**	**MV_Y0_1_L1 (B)**	**MV_Y0_1_L1**
	15:0	**Duplicate (A)**	**Duplicate (A)**	**MV_Y0_1_L0 (B)**	**MV_Y0_1_L0**
W1.2	31:16	**Duplicate (A)**	**MV_Y0_2_L1 (B)**	**Duplicate (A)**	**MV_Y0_2_L1**
	15:0	**Duplicate (A)**	**MV_Y0_2_L0 (B)**	**Duplicate (A)**	**MV_Y0_2_L0**
W1.3	31:16	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y0_3_L0**
	15:0	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y0_3_L1**

Packed-Size Motion Vector Data Block

In the packed case, no redundant motion vectors are sent. So the number of motion vectors sent, as specified by **MvQuantity** is the same as the motion vectors that will be packed (**MvPacked**).

The following tables are for information only. Fields like MvQuantity and MvPacked are not required interface fields.

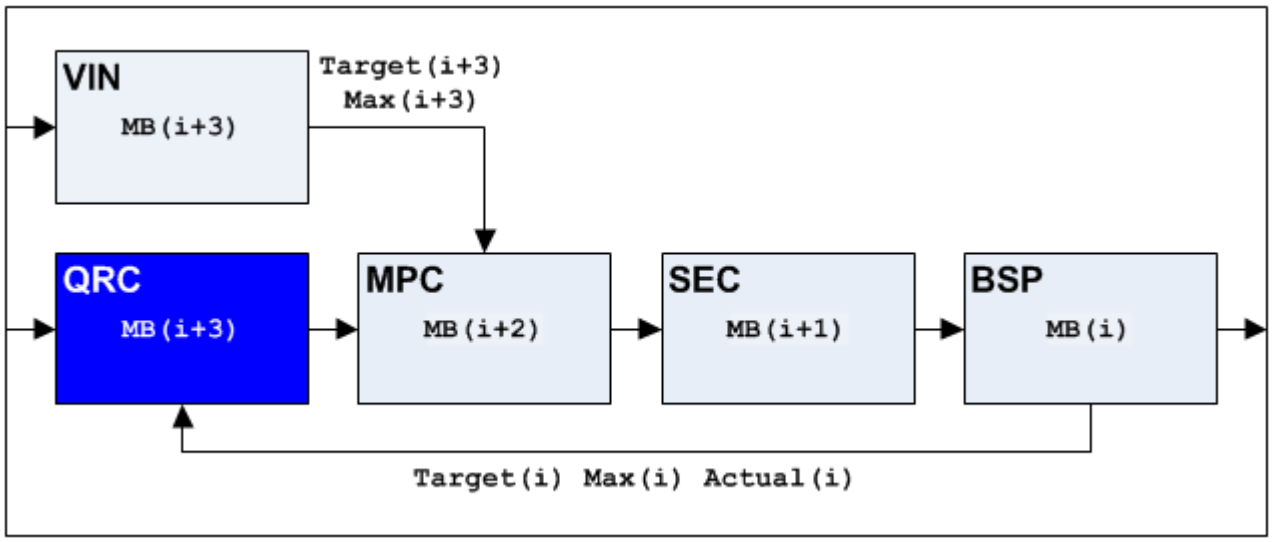
MbSkipFlag	MbType	Description	Mv Quantity	MvSize	(Minimal MvSize)
1	1	P_Skip_16x16	0	8	1
0	1	BP_L0_16x16	1	8	1
0	2	B_L1_16x16	1	8	1
0	3	B_Bi_16x16	2	8	2
0	4	BP_L0_L0_16x8	2	8	4
0	5	BP_L0_L0_8x16	2	8	4
0	6	B_L1_L1_16x8	2	8	8
0	7	B_L1_L1_8x16	2	8	8
0	8	B_L0_L1_16x8	2	8	8
0	9	B_L0_L1_8x16	2	8	8
0	0Ah	B_L1_L0_16x8	2	8	8
0	0Bh	B_L1_L0_8x16	2	8	8
0	0Ch	B_L0_Bi_16x8	3	8	8
0	0Dh	B_L0_Bi_8x16	3	8	8
0	0Eh	B_L1_Bi_16x8	3	8	8
0	0Fh	B_L1_Bi_8x16	3	8	8
0	10h	B_Bi_L0_16x8	3	8	8
0	11h	B_Bi_L0_8x16	3	8	8
0	12h	B_Bi_L1_16x8	3	8	8
0	13h	B_Bi_L1_8x16	3	8	8
0	14h	B_Bi_Bi_16x8	4	8	8
0	15h	B_Bi_Bi_8x16	4	8	8
0	16h	**BP**_8x8	[3]4	8 or 32	8 or 32

When MbType = 22, BP_8x8, take the sum of four individual 8x8 subblocks

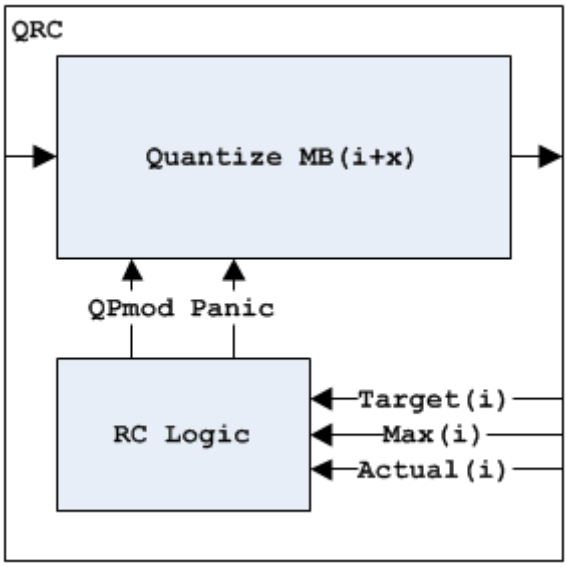
Direct8x8Pattern	SubMb Shape	SubMb PredMode	Description	Mv Quantity	Mv Size	(Min MvSize)
OR	OR	OR		ADD	ADD	ADD
1	0	0	P_Skip_8x8 B_Direct_L0_8x8 (B-Skip_ L0_8x8)	0	2	1
1	0	1	B_Direct_L1_8x8 (B-Skip_ L1_8x8)	0	2	1
1	0	2	B_Direct_Bi_8x8 (B-Skip_ Bi_8x8)	0	2	2
1	3	0	P_Skip_4x4 B_Direct_L0_4x4 (B-Skip_ L0_4x4)	0	8	4
1	3	1	B_Direct_L1_4x4 (B-Skip_ L1_4x4)	0	8	4
1	3	2	B_Direct_Bi_4x4 (B-Skip_ Bi_4x4)	0	8	8
0	0	0	BP_L0_8x8	1	2	1
0	0	1	B_L1_8x8	1	2	1
0	0	2	B_BI_8x8	2	2	2
0	1	0	BP_L0_8x4	2	8	4
0	1	1	B_L1_8x4	2	8	4
0	1	2	B_BI_8x4	4	8	8
0	2	0	BP_L0_4x8	2	8	4
0	2	1	B_L1_4x8	2	8	4
0	2	2	B_BI_4x8	4	8	8
0	3	0	BP_L0_4x4	4	8	4
0	3	1	B_L1_4x4	4	8	4
0	3	2	B_BI_4x4	8	8	8

Macroblock Level Rate Control

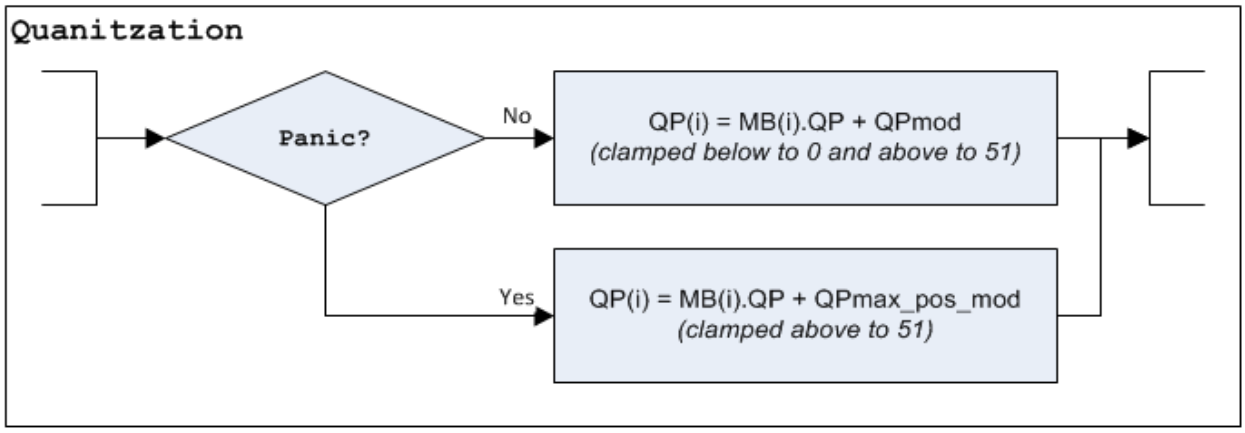
The QRC (Quantization Rate Control) unit receives data from BSP (Bit Serial Packer) and VIN (Video In) and generates adjustments to QP values across macroblocks.



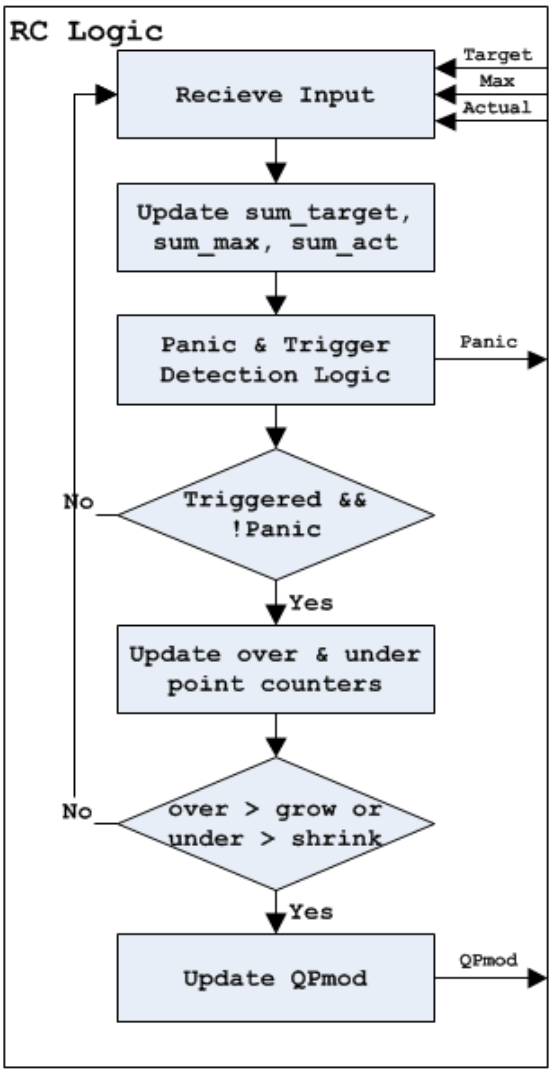
QRC can be logically partitioned into two units as shown below.



Macroblock level rate control is handled by the RC logic and the quantization logic.



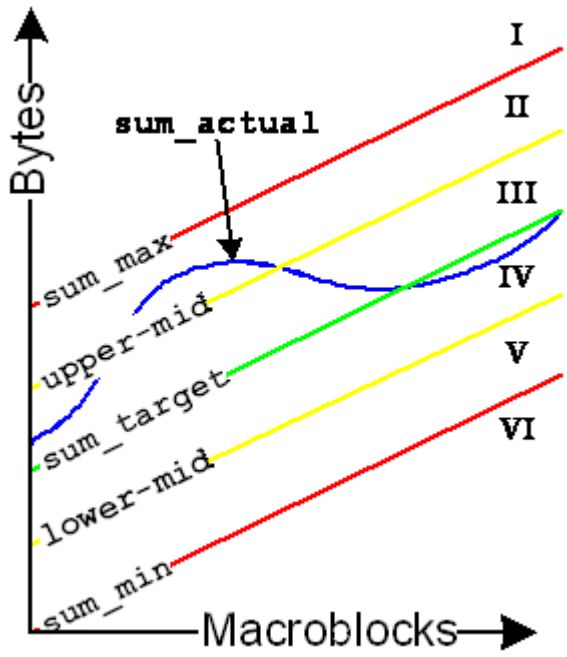
The signals QPmod and panic are generated by the RC logic based on data feedback from BSP. A flowchart of the RC logic is given below.



Theory of Operation Overview

BSP will generate a byte estimate for each macroblock packed. Additionally, the user will specify a target and max size per macroblock. The running sum of these signals (actual, target, max) creates “curves” which are used to identify when QP adjustments are necessary (see figure below). Three more curves are symmetrically generated by QRC (upper_midpt, lower_midpt, sum_min) from target and max. The values of target and max are specified by the user will dictate the shape of these curves.

The difference between sum_actual and sum_target (called ‘bytediff’) identifies the margin of error between the target and actual sizes. The difference between the current bytediff and the previously calculated bytediff represents the rate of change in this margin over time. The sign of this rate is used to identify if the correction is trending in the appropriate direction (towards bytediff = 0).



QPmod

Each macroblock will have a requested QP (which could vary across macroblocks or remain constant). QPmod is to be added to the QP requested. QPmod will be positive when the target was under-predicted and negative when the target is over-predicted.

QPmod is incremented or decremented when internal counters (called ‘over’ and ‘under’) reach tripping points (called ‘grow’ and ‘shrink’). For each MB processed and based on which region (1-6) sum_actual falls in, various amounts of points are added to either counters. If over exceeds grow, QPmod is incremented whereas if under exceeds shrink, QPmod is decremented.

To dampen the effect of repeated changes in the same direction, an increase in resistance for that direction and decrease in resistance for the complementary direction occurs (called ‘grow_resistance’ and

'shrink_resistance'). This resistance is added to grow or shrink, which then requires more points to trip the next correction in that direction.

The user can specify guard-bands that limit the amount QPmod can be modified. QPmod cannot exceed QPmax_pos_mod or become less than -QPmax_neg_mod_abs.

Triggering

The RC unit begins to modify QPmod occurs only when it is triggered.

Three levels of triggering exist: always, gentle, loose. Always means that RC will be active once sum_actual reaches regions 3 or 4. Gentle will trigger RC once sum_actual reaches regions 2 or 5. Loose waits to trigger RC when sum_actual reaches regions 1 or 6.

RC will deactivate (triggered = false) once sum_actual begins to track sum_target over a series of macroblocks. Specifically, the sign of the rate of change for bytediff is monitored over a window of macroblocks. When the sum of these signs over the window falls within a tolerance value (called 'stable'), triggered will reset to false.

Panic

When enabled, panic mode will occur whenever sum_actual reaches region 1 and will remain so until sum_actual reaches region 4. When panicking, all macroblocks will be quantized with QP = MB(n).QP + QPmax_pos_mod, clamped to 51.

User Controls

This unit achieves a large flexibility by allowing the user to define various key parameters. At the per-macroblock level, the values of target and max are specified and will create various shapes of curves that sum_actual will be compared against.

Per-slice, the user can specify the triggering sensitivity and the tolerance required to disable the trigger. Additionally, the user can enable panic detection.

The point values assigned to each of the 6 regions are exposed to the user which allow for asymmetrical control for over and under predictions amongst other things. Additionally, the user can specify the initial values of grow and shrink along with the resistance values applied when correction is invoked.

Lastly, the maximum and minimum values for QPmod are also exposed to the user.

AVC Encoder MBAFF Support

1. **Algorithm**

   Prediction of current macroblock motion vector is possible from neighboring macroblocks mbAddrA/mbAddrD/mbAddrB/mbAddrC/mbAddrA+1/mbAddrD+1/mbAddrB+1/mbAddrC+1. The selection of these macroblocks depends on coding type(field/frame) of current macroblock pair and the coding of neighboring macroblock pair.

   Selection of these macroblock pairs is described in detail in following sections.

1.1 Selection of Top LeftMB pair: The selection of Top Left MB pair depends on coding type of current and also top left macroblock pair.

1.2 Selection of LeftMB pair: The selection of Left MB pair depends on coding type of current and also left macroblock pair.

1.3 Selection of Top MB pair: The selection of Top MB pair depends on coding type of current and also top macroblock pair.

1.4 Selection of Top RightMB pair: The selection of Top Right MB pair depends on coding type of current and also top right macroblock pair.

1.5 Motion Vector and refIdx Scaling: Motion vectors and reference index of neighboring macroblocks (mbAddrA/mbAddrB/mbAddrC/mbAddrD) should be scaled before using them into prediction equations. Again the scaling depends on coding type of current and neighboring macroblock pair which is described as follows,

- If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock ...

```
mvLXN[ 1 ] = mvLXN[ 1 ] / 2 (8-214)
refIdxLXN = refIdxLXN * 2   (8-215)
```

- Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock ...

```
mvLXN[ 1 ] = mvLXN[ 1 ] * 2 (8-216)
refIdxLXN = refIdxLXN / 2   (8-217)
```

- Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

MPEG-2

MPEG2 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_MPEG2_PIC_STATE

MPEG2 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_MPEG2_BSD_OBJECT

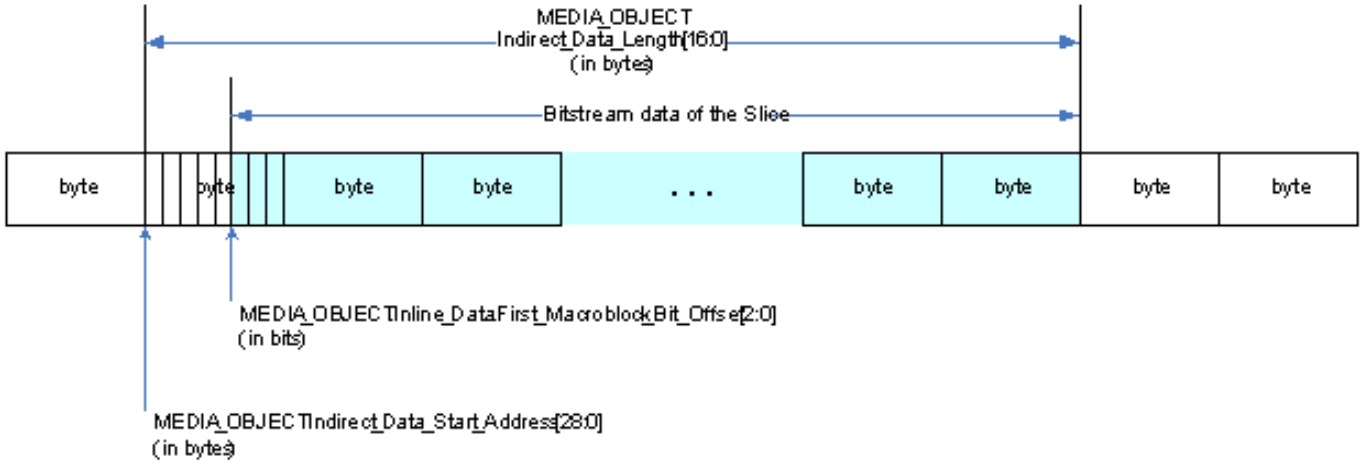
MFD_MPEG2_BSD_OBJECT Inline Data Description

Indirect Data Description

The indirect data start address in MFD_MPEG2_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the slice header. It provides the byte address for the first macroblock of the slice. Together with the First Macroblock Bit Offset field in the inline data, it provides the bit location of the macroblock within the compressed bitstream.

The indirect data length in MFD_MPEG2_BSD_OBJECT provides the length in bytes of the bitstream data for this slice. It includes the first byte of the first macroblock and the last **non-zero** byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte. *Indirect Data Description* illustrates these parameters for a slice data.

Indirect data buffer for a slice



MPEG2 Encoder PAK Commands

The MFC_MPEG2_PAK_INSERT_OBJECT Command is identical to the MFC_AVC_PAK_INSERT_OBJECT command as described in this document.

The MFC_MPEG2_STITCH_OBJECT Command is identical as MFC_AVC_STITCH_OBJECT command as described in this document.

MFC_MPEG2_SLICEGROUP_STATE

MFC_MPEG2_PAK_OBJECT

PAK Object Inline Data Description – MPEG-2

The Inline Data includes all the required MB encoding states, constitute part of the Slice Data syntax elements, MB Header syntax elements and their derivatives. It provides information for the following operations:

1. Forward and Inverse Transform
2. Forward and Inverse Quantization
3. Advanced Rate Control (QRC)
4. MB Parameter Construction (MPC)
5. VLC encoding
6. Bit stream packing
7. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_MPEG2_PAK_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_MPEG2_PAK_OBJECT command.

The inline data has been designed to match AVC MB structure for efficient transcoding.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

DWord	Bit	Description
1	31:27	Reserved: MBZ
	22-20	**MvFormat (Motion Vector Size)**. This field specifies the size and format of the input motion vectors. This field is reserved (MBZ) when the **IntraMbFlag** = 1. The valid encodings are: **011 = Unpacked: Two motion vector pairs** Others are reserved. *(The following encodings are intended for other formats:* *001 = 1MV: one 16x16 motion vector* *010 = 2MV: One 16x16 motion vector pair* *011 = 4MV: Four 8x8 motion vectors, or Two 16x8 motion vector pairs* *100 = 8MV: Four 8x8 motion vector pairs* *101 = 16MV: 16 4x4 motion vectors* *110 = 32MV: 16 4x4 motion vector pairs* *111 = Packed, number of MVs is given by **packedMvNum**.)*
	19	**CbpDcY.** This field specifies if the Luma DC coded. Must be 1 for MPEG-2.
	18	**CbpDcU.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.**
	17	**CbpDcV.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.**
	16	Reserved: MBZ
	15	**TransformFlag** Used to indicate transformation type for MPEG-2. 0 = Frame DCT transformation 1 = Field DCT transformation
	14	**FieldMbFlag** For MPEG-2, this flag is set to 1 if either the picture is in field type or the MB is INTER of field type, i.e. split into two 16x8 field blocks.

	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	**MbType** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows an unified encoding for inter and intra macroblocks according to MFX Encoding reference as shown in Figure A.
	7:3	Reserved : MBZ
	2	**SkipMbFlag** By setting it to 1, this field forces an inter macroblock to be encoded as a skipped macroblock. It is equivalent to mb_skip_flag in AVS spec, Hardware honors input MVs for motion prediction and forces CBP to zero. By setting it to 0, an inter macroblock will be coded as a normal inter macroblock. The macroblock may still be coded as a skipped macroblock, according to the macroblock type conversion rules described in the later sub sections. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. Hardware ignores this field for an intra macroblock. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock **Note:** When this flag is set to 1, the correct MVs are assumed for HW decoder to generate decoded reconstruction frame.
	1:0	**InterMbMode** This field is provided to carry redundant information as that encoded in MbType. This field is only valid if **IntraMbFlag** =0, otherwise, it is ignored by hardware.
2	31:16	**MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U16 in unit of macroblock.

	15:0	**MbXCnt (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U16 in unit of macroblock.
3	31:24	**MaxSizeInWord** PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.
	23:16	**TargetSizeInWord** PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients.
	15:13	**MBZ**
	12:0	**Cbp –** Coded Block Pattern. This field specifies whether blocks are present or not. Format = 6-bit mask (or 8-bit, & 12-bit, for 422 and 444). Bit 11: Y0Bit 10: Y1Bit 9: Y2Bit 8: Y3 Bit 7: Cb4Bit 6: Cr5Bits 0-5: MBZ
4	31	**LastMbInSlice –** the last MB in a slice.
	30	**FirstMbInSlice –** the first slice in a slice, it requires slice header insertion.
	29:28	**MBZ**
	27	**EnableCoeffClamp** 1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization 0 = no clamping
	26	**LastMbInSG** 1 – the current MB is the last MB in the current slice group.
	25	**MbSkipConvDisable** This is a per-MB level control to enable and disable skip conversion. This field is ORed with SkipConvDisable field. This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in *Macroblock Type Conversion Rules* 0 - Enable skip type conversion for the current macroblock 1 – Disable skip type conversion for the current macroblock

<table>
<tr><td></td><td>24</td><td>FirstMbInSG
1 – the current MB is the last MB in the current slice group.</td></tr>
<tr><td></td><td>23:20</td><td>MBZ</td></tr>
<tr><td></td><td>19:16</td><td>MvFieldSelect – Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the ISO/IEC 13818-2 (see also §7.6.4).
<table>
<tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>16</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>17</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>18</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>19</td><td>1</td><td>1</td><td>3</td></tr>
</table>
Format = MC_MotionVerticalFieldSelect.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>15:5</td><td>MBZ Reserved</td></tr>
<tr><td></td><td>4:0</td><td>QpScaleCode</td></tr>
<tr><td>5</td><td>31:16</td><td>MV[0][0].y – the y coordinate of the first forward MV
if Mv[0][0] n/a:
if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].
else it MUST be set to the value 0</td></tr>
<tr><td></td><td>15:0</td><td>MV[0][0].x – the x coordinate of the first forward MV
if Mv[0][0] n/a:
if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].
else it MUST be set to the value 0</td></tr>
<tr><td>6</td><td>31:0</td><td>MV[1][0] – the first backward MV
if Mv[1][0] n/a: it MUST be set to the same value as Mv[0][0]</td></tr>
</table>

7	31:0	**MV[0][1]** – the second forward MV if Mv[0][1] n/a: if Mv[1][1] available, it MUST be set to the same value as Mv[1][1]. else it MUST be set to the same value as Mv[0][0]
8	31:0	**MV[1][1]** – the second backward MV if Mv[1][1] n/a: it MUST be set to the same value as Mv[1][0]

The mapping between MPEG-2 spec and MfxMbCode can be achieved according to the following:

1) Renamed variables with identical meaning:

MPEG-2 Spec	MFX API	Value
macroblock_quant	**MbQuantPresent**	0 or 1
macroblock_intra	**IntraMbFlag**	0 or 1
dct_type	**Transform8x8Flag**	0 or 1
macroblock_pattern	**Cbp8x8**	remapped

2) Macroblock type remapping:

		B-spec Entry					MPEG-2 Spec				
Frame Type	Mb Type	Intra Mb Flag	Skip Mb Flag	Mb Type 5Bits	Field Mb Flag	Inter Mb Mode	macroblock_intra	motion_type_bit0	motion_type_bit1	motion_forward	motion_backward
IPB	Intra	1	0	1Ah	0/1	-	1	-	-	-	-
P **B** B	Skip	0	1	01h 02h 03h	0/1	0	0	-	-	1 0 1	0 1 1
P	0-MV*	0	0	01h	0/1	0	0	-	-	0	0

		B-spec Entry					MPEG-2 Spec				
Frame Type	Mb Type	Intra Mb Flag	Skip Mb Flag	Mb Type 5Bits	Field Mb Flag	Inter Mb Mode	macroblock_intra	motion_type_bit0	motion_type_bit1	motion_forward	motion_backward
P Frame	Frame type	0	0	01h	0	0	0	0	1	1	0
P Frame	Field type	0	0	04h	1	1	0	1	0	1	0
P Frame	dual prime	0	0	19h	0	0	0	1	1	1	0
P Field	One 16x16	0	0	01h	1	0	0	1	0	1	0
P Field	Two 16x8	0	0	04h	1	1	0	0	1	1	0
P Field	dual prime	0	0	19h	1	0	0	1	1	1	0
B Frame	Frame type	0	0	01h 02h 03h	0	0	0	0	1	1 0 1	0 1 1
B Frame	Field type	0	0	04h 06h 14h	1	1	0	1	0	1 0 1	0 1 1

		B-spec Entry					MPEG-2 Spec				
Frame Type	Mb Type	Intra Mb Flag	Skip Mb Flag	Mb Type 5Bits	Field Mb Flag	Inter Mb Mode	macroblock_intra	motion_type_bit0	motion_type_bit1	motion_forward	motion_backward
B Field	One 16x16	0	0	01h 02h 03h	1	0	0	1	0	1 0 1	0 1 1
B Field	Two 16x8	0	0	04h 06h 14h	1	1	0	0	1	1 0 1	0 1 1

- Notice that there is no special way to indicate 0 motion vector case for P frame. It is for PAK to handle internally by checking up the motion vector values.
- Notice also, the MbType5bits is adapted from AVC DXVA macroblock types. It may seems awkward from MPEG-2 perspective, but provides a common VME interface for us for simpler HW design and help the advanced transcoding solution.

MFX HW Interface and DXVA Conversion

Map DXVA to HW BSpec

Location		HW	
		BSPEC	
BYTE	Dword	MPEG-2	DXVA
DW0			
0		**MbMode**	
0.0-1	0[0-1]	InterMbMode	see (A)
0.2	0[2]	SkipMbFlag	<-MBskipsFollowing
0.3	0[3]	mbz	
0.4-0.5	0[4-5]	IntraMbMode	IntraMacroblock
0.6	0[6]	mbz	
0.7	0[7]	FieldMbPolarity	derived
1		**MbType**	

Location		HW	
		BSPEC	
BYTE	Dword	MPEG-2	DXVA
1.0-1.4	0[8-12]	MbType5Bits	see (A)
1.5	0[13]	IntraMbFlag	IntraMacroblock
1.6	0[14]	FieldMbFlag	see (A)
1.7	0[15]	TransformFlag	FieldResidual
2		MbFlag	
2.0	0[16]	ResidDataFlag	HostResDiff
2.1	0[17]	CbpDcV	PAK control
2.2	0[18]	CbpDcU	PAK control
2.3	0[19]	CbpDcY	PAK control
2.4-2.6	0[20-22]	MvFormat	= 3, derived
2.7	0[23]	mbz	
3	0[24-31]	PackedMvNum	see (A)
DW1			
4-5	1[0-15]	MbXCnt	wMBaddress
6-7	1[16-31]	MbYCnt	wMBaddress
DW2			
8	2[0-7]		bNumCoef[0]
8.0-8.5	2[0-5]	mbz	
8.6-8.7	2[6-7]	CbpAcUV	PAK control
9	2[8-11]	CbpAcY	PAK control
	2[12-15]	mbz	
10	2[16-23]	TargetedSzInWord	
11	2[24-31]	MaxSzInWord	
DW3			
12		Qscale	derived
12.0-6	3[0-6]	QScaleCode	
12.7	3[7]	QScaleType	
13	3[8-15]	mbz	
14	3[16-19]	MvFieldSelect	MvertFieldSel
	3[20-23]	mbz	
15		MbExtFlag	
15.0	3[24]	mbz	
15.1	3[25]	SkipMvConvDisable	
15.2	3[26]	LastMbFlag	PAK control

Location		HW	
		BSPEC	
BYTE	Dword	MPEG-2	DXVA
15.3	3[27]	EnableCoeffClamp	PAK control
15.4-5	3[28-29]	MbScanMethod	MBscanMethod
15.6	3[30]	NewSliceFlag	PAK control
15.7	3[31]	EndSliceFlag	PAK control
DW4-7			
16-32	4-7[all]	MV[2][2][2]	MVector[4][2]

(A): Set **InterMbMode, MbType5bits, FieldMbFlag**, and **PackedMvNum** from DXVA parameters:

```
 if(IntraMacroblock) return (TYPE_INTRA);
 else if(MotionType==3){ // dual prime
    MbType5bits = 0x19; FieldMbFlag = 0; InterMbMode = 0; PackedMvNum = 2;     return
(DUAL_PRIME);
 }
 else{
    IsFieldFrame = a PicState derivative;     switch(MotionType+IsFieldFrame{
       case 1: // Two 16x8 field in Frame Frame
       case 3: // Two 16x8 field in Field Frame
          FieldMbFlag = 1; InterMbMode= 1;          switch(MotionForward |Motionbackward «1)){
             case 1:
                MbType5bits = 4; PackedMvNum = 2;                 break;
             case 2:
                MbType5bits = 6; PackedMvNum = 2;                 break;
             case 3:
                MbType5bits = 0x14; PackedMvNum = 4;                 break;
          }
          break;
       case 2: // 16x16 block in either case
          FieldMbFlag = IsFieldFrame; InterMbMode = 0;
switch(MotionForward|(Motionbackward«1)){
             case 1:
                MbType5bits = 1; PackedMvNum = 1;                 break;
             case 2:
                MbType5bits = 2; PackedMvNum = 1;                 break;
             case 3:
                MbType5bits = 3; PackedMvNum = 2;                 break;
          }
          break;
    }
 }
```

Map HW Bspec to DXVA

Location	BSPEC	
BYTE	DXVA	MPEG-2
0-1	wMBaddress	= MbYCnt*MbW + MbXCnt
2-3	wMBtype	
2.0	IntraMacroblock	= IntraMbFlag

Location	BSPEC	
BYTE	DXVA	MPEG-2
2.1	MotionForward	see (B)
2.2	MotionBackward	see (B)
2.3	Motion4MV	VC-1 only, MBZ for Mpeg-2
2.4	Reserved	
2.5	FieldResidual	= **TranformFlag**
2.6-2.7	MBscanMethod	= **MbScanMethod**
3.0-3.1	MotionType	see (B)
3.2	HostResDiff	= **ResidDataFlag**
3.3	Reserved	
3.4-3.7	MvertFieldSel	= **MvFieldSelect**
4	MBskipsFollowing	count **SkipMbFlag**
5-7	MBdataLocation	n/a
8-9	wPatternCode	= **CbpAcY|UV**
10-15	bNumCoef[6]	n/a
16-32	MVector[4][2]	= **MV[2][2][2]**

(B): Set **MBtype** and **MotionType** from Bspec interface

```
if(MbIntraFlag) return (TYPE_INTRA);
else {
    if(MbType5Bits&8) { // dual prime
        MotionForward  = 1;
        MotionBackward = 0;
        MotionType     = 3;
        return (DUAL_PRIME);
    }
    else {
        // redundant: InterMbMode = !!(MbType5Bits&4);
        if(InterMbMode) {
            MotionForward  = !(MbType5Bits&2);
            MotionBackward = !!(MbType5Bits&0x12);
        }
        else {
            MotionForward  = (MbType5Bits&1);
            MotionBackward = !!(MbType5Bits&2);
        }
        MotionType = 2-(InterMbMode^FieldMbFlag);

        // equivalently the 2 bits are:
        // MotionType0 = (InterMbMode^FieldMbFlag);
        // MotionType1 = ~MotionType0;

        return (TYPE_INTER);
    }
}
```

Video Codec VC-1

This section describes support for the open video compression standard VC-1, which is the common name for SMPTE 421M approved on April 3, 2006.

VC1 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_DIRECTMODE_STATE

VC1 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_VC1_LONG_PIC_STATE

AltPQuantConfig and **AltPQuantEdgeMask** are derived based on the following variables: *DQUANT*, *DQUANTFRM*, *DQPROFILE*, *DQSBEDGE*, *DQDBEDGE*, and *DQBILEVEL* defined in the VC1 standard, as shown in the following table.

Definition of AltPQuantConfig and AltPQuantEdgeMask

Inputs						Outputs		
DQUANT	DQUANT FRM	DQ PROFILE	DQDB EDGE	DQSB EDGE	DQBI LEVEL	AltPQuant Config	AltPQuant EdgeMask	Description
0	-	-	-	-	-	00b	0000b	No **AltPQuant**
1	0	-	-	-	-	00b	0000b	No **AltPQuant**
1	1	11b	-	-	0	10b	0000b	All MBs are different with *MQDIFF* and *ABSMQ*
1	1	11b	-	-	1	11b	0000b	All MBs may switch with 1-bit *MQDIFF*
2	-	-	-	-	-	01b	1111b	All edge MBs
1	1	00b	-	-	-	01b	1111b	All edge MBs
1	1	01b	00b	-	-	01b	0011b	Left and top MBs
1	1	01b	01b	-	-	01b	0110b	Top and right MBs
1	1	01b	10b	-	-	01b	1100b	Right and bottom MBs

Inputs						Outputs		
DQUANT	DQUANT FRM	DQ PROFILE	DQDB EDGE	DQSB EDGE	DQBI LEVEL	AltPQuant Config	AltPQuant EdgeMask	Description
1	1	01b	11b	-	-	01b	1001b	Bottom and left MBs
1	1	10b	-	00b	-	01b	0001b	Left MBs
1	1	10b	-	01b	-	01b	0010b	Top MBs
1	1	10b	-	10b	-	01b	0100b	Right MBs
1	1	10b	-	11b	-	01b	1000b	Bottom MBs

MFD_VC1_SHORT_PIC_STATE

Intel HW does not use the MVMODE and MVMODE2 provided at the revised DXVA2 VC1 VLD interface, instead, HW will decode them directly from the bitstream picture header.

MFD_VC1_BSD_OBJECT

For VC1, a slice/picture is always started with MB x positon equal to 0. Hence, no need to include in the Object Command.

Handling Emulation Bytes

In general, VC1 BSD unit is capable of handling emulation prevention bytes. However, there is a corner case that requires host software's intervention. Host software needs to overwrite the emulation byte if it overlaps the macroblock layer decode and there is not enough information for the hardware to detect the emulation byte.

The emulation bytes might have an overlap between the picture states and the first macroblock data. If the emulation bytes are 0x00 **0x000x03** 0x00 and the macroblock data starts in the middle of byte1 (**0x00**), then the host software needs to overwrite the **0x03** byte location with the previous byte (**0x00**) and change the byte offset accordingly. The hardware wouldn't know what the 1st byte was and will miss this **0x03** removal.

VP8

MFX_VP8_STATISTICS - Encoder Only

Address offset			Name	Description	
0	31:0	32-bit	P0 Partition Bit Size	// Raw bit count per Partition	No Fulsim Validation
1	31:0	32-bit	P1 Partition Bit Size		No Fulsim Validation
2	31:0	32-bit	P2 Partition Bit Size		No Fulsim Validation
3	31:0	32-bit	P3 Partition Bit Size		No Fulsim Validation
4	31:0	32-bit	P4 Partition Bit Size		No Fulsim Validation
5	31:0	32-bit	P5 Partition Bit Size		No Fulsim Validation
6	31:0	32-bit	P6 Partition Bit Size		No Fulsim Validation
7	31:0	32-bit	P7 Partition Bit Size		No Fulsim Validation
8	31:0	32-bit	P8 Partition Bit Size		No Fulsim Validation
9	31:0	32-bit	P1-8 Partition Bit Size Sum	// This is raw BIT sum of Partition1-8, Not Byte sum.	No Fulsim Validation
					No Fulsim Validation
10	31:0	32-bit	Segment0 Total Bit Count	// Raw bit count per Segment	No Fulsim Validation
11	31:0	32-bit	Segment1 Total Bit Count		No Fulsim Validation
12	31:0	32-bit	Segment2 Total Bit Count		No Fulsim Validation
13	31:0	32-bit	Segment3 Total Bit Count		No Fulsim Validation
14	15:0	16-bit	Segment0 Num of MB	// Num of MB per Segment	No Fulsim Validation
	31:16	16-bit	Segment1 Num of MB		No Fulsim Validation

15	15:0	16-bit	Segment2 Num of MB		No Fulsim Validation
	31:16	16-bit	Segment3 Num of MB		No Fulsim Validation
16	31:0	32-bit	P0 Partition Bit Size before CPBAC	// Bin Count of Syntax Element before CPBAC	No Fulsim Validation
17	31:0	32-bit	P1 Partition Bit Size before CPBAC		No Fulsim Validation
18	31:0	32-bit	P2 Partition Bit Size before CPBAC		No Fulsim Validation
19	31:0	32-bit	P3 Partition Bit Size before CPBAC		No Fulsim Validation
20	31:0	32-bit	P4 Partition Bit Size before CPBAC		No Fulsim Validation
21	31:0	32-bit	P5 Partition Bit Size before CPBAC		No Fulsim Validation
22	31:0	32-bit	P6 Partition Bit Size before CPBAC		No Fulsim Validation
23	31:0	32-bit	P7 Partition Bit Size before CPBAC		No Fulsim Validation
24	31:0	32-bit	P8 Partition Bit Size before CPBAC		No Fulsim Validation
25	31:0	32-bit	Reserved		
26	31:0	32-bit	Reserved		
27	31:0	32-bit	Reserved		
28	31:0	32-bit	Reserved		
29	31:0	32-bit	Reserved		
30	31:0	32-bit	Reserved		
31	31:0	32-bit	Reserved		
32	31:0	32-bit	mb skip prob Total cnt	Total Number of MB when MBNoCoeffSkip == 1(if MBNoCoeffSkip == 0, this field =0) Please see foot notes 1.	
33	31:0	32-bit	mb skip prob cnt	Number of MB with MBSkip == 1 when MBNoCoeffSkip == 1. Please see foot notes 1.	
303-34	15:0		token_statistic count	Token Statistics Counters. Please see foot notes 2.	
	31:16	Reserved			

Programming Note	
Context:	MFX_VP8_STATISTICS - Encoder Only

1. MBSkip_Prob_Total_Cnt and MBSkip_Prob_Cnt are collected to generate MBSkipProbability. After bit packing, Hardware returns both MBSkip_Prob_Total_Cnt and MBSkip_Prob_Cnt. Optimal packing could be performed in subsequent pass using MBSkipProbability of

   round (256 * (MBSkip_Prob_Total_Cnt- MBSkip_Prob_Cnt) / MBSkip_Prob_Total_Cnt))

2. Token Statistics counters collects token statistics of particular plane (4), coefficient band(8), neighbor context(3) and tree node position(11) as described in WebM Spec. Out of the space of 1056 counters, there are 270 of which has high significant in compression efficiency and are chosen for statistics collection.

VP8 Common Commands

Following are VP8 Common Commands:

MFX_VP8_PIC_STATE

For VP8 HW PAK, there are four VP8 versions supported and their programming is shown in Table1 below.

Version	MC Filter Select	Chroma Full Pixel MC Filter Mode	DBLK FilterType	DBLK FilterLevel for Segment0
0	0	0	0	Any FilterLevel
1	1	0	1	Any FilterLevel
2	1	0	0	0
3	1	1	1	0

Table1: VP8 Version

MC Filter Select: MFX_VP8_PIC_STATE.DW2.Bit0

Chroma Full Pixel MC Filter Mode: MFX_VP8_PIC_STATE.DW2.Bit1

DBKL Filter Type: MFX_VP8_PIC_STATE.DW2.Bit4

DBLK Filter Level for Segment0: MFX_VP8_PIC_STATE.DW3.Bit5:0

1. Note that when multiple segment is enabled, if Segment0 DBLK Filter is programmed to 0, Segment1,2,3 DBLK Filter should be set to 0 as well.
2. Note that MFX_VP8_Encoder_CFG.BW22.Bit22:20 (Bitstream Format Version). This field is used for generating Uncompressed header only. It is not used to control any Filter.

VP8 Decoder Commands

Following are VP* Decoder Commands:

MFD_VP8_BSD_OBJECT

VP8 Encoder Commands

MFX_VP8_Encoder_CFG

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

MFX_VP8_PAK_OBJECT

VP8 PAK Object inline data:

Inline Data Description - VP8 PAK OBJECT

Y_Mode for macroblock in non-B mode

0	DC_PRED
1	V_PRED
2	H_PRED
3	TM_PRED
4	B_PRED
5	NEARESTMV
6	NEARMV
7	ZEROMV
8	NEWMV
9	SPLITMV

2 B mode

0	B_DC_PRED
1	B_TM_PRED
2	B_VE_PRED
3	B_HE_PRED
4	B_LD_PRED
5	B_RD_PRED
6	B_VR_PRED
7	B_VL_PRED
8	B_HD_PRED

9	B_HU_PRED
10	SPLIT_LEFT
11	SPLIT_ABOVE
12	SPLIT_ZERO
13	SPLIT_NEW

JPEG and MJPEG

JPEG Decoder Commands

Following are JPEG Decoder Commands:

MFD_JPEG_BSD_OBJECT

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.

MFX_JPEG_PIC_STATE

For JPEG decoding, the following program note is informative.

For **Rotation**, it is important to note that rotation of 90 or 270 degrees also requires exchanging **FrameWidthInBlksMinus1** with **FrameHeightInBlksMinus1** in the command. In addition, the rotation of 90 or 270 degrees also requires transportation of the quantization matrix will be transposed into the position (y, x).

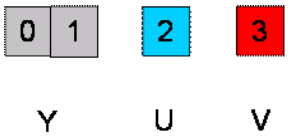
Chroma type is determined by the values of horizontal and vertical sampling factors of the components (*Hi* and Vi where *i* is a component id) in the Frame header as shown in the following table.

	H1	H2	H3	V1	V2	V3
0: YUV400	r	Not available	Not available	r	Not available	Not available
1: YUV420	2	1	1	2	1	1
2: YUV422H_2Y	2	1	1	1	1	1
3: YUV444	1	1	1	1	1	1
4: YUV411	4	1	1	1	1	1
5: YUV422V_2Y	1	1	1	2	1	1
6: YUV422H_4Y	2	1	1	2	2	2
7: YUV422V_4Y	2	2	2	2	1	1

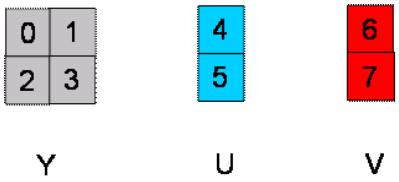
For YUV400, the value of *V1* can be 1, 2, or 3 and will be same as the value of *H1*, and the Minimum coded unit (MCU) is one 8x8 block. For the other chroma formats, if non-interleaved data, the MCU is one 8x8 block. For interleaved data, the MCU is the sequence of block units defined by the sampling factors of the components.

For example, the following figures show the MCU structures of interleaved data and the decoding order of blocks in the MCU:

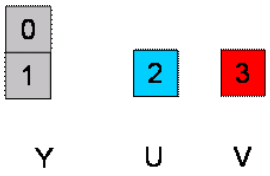
422H_2Y



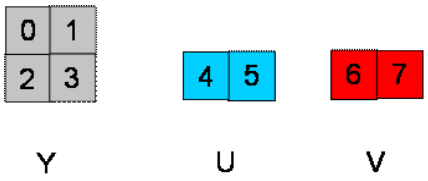
422H_4Y



422V_2Y

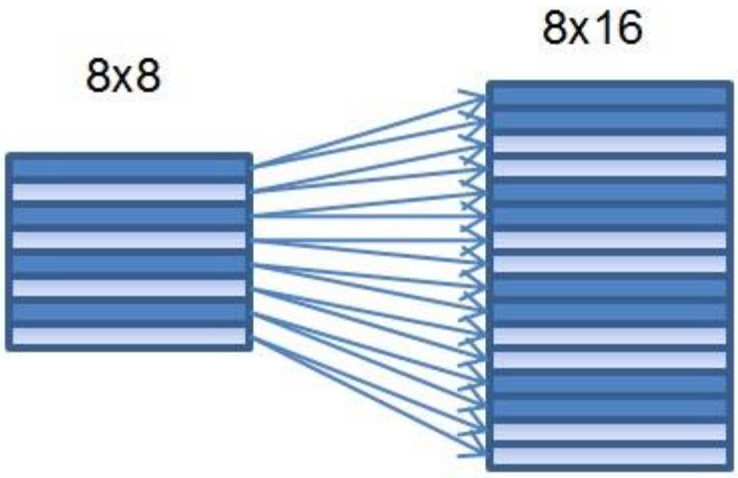


422V_4Y

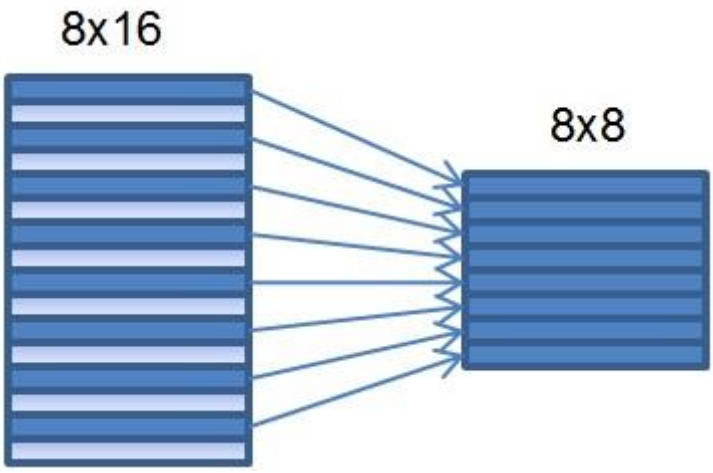


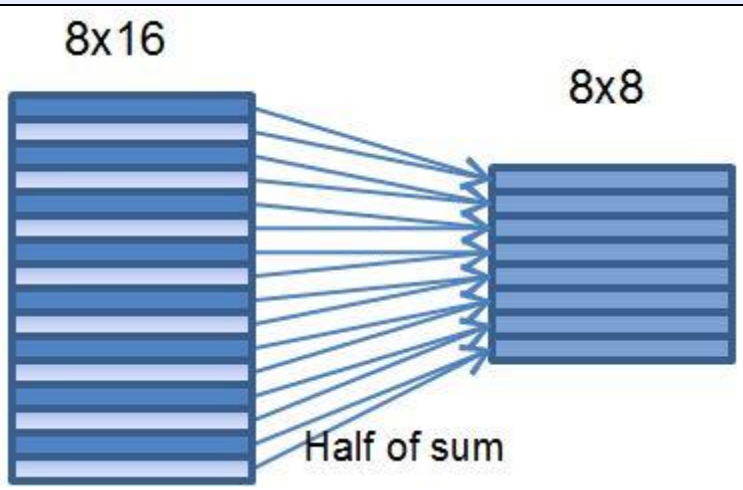
If picture width *X* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the decoding process needs to extend the number of samples by one or more additional blocks so that the number of blocks is an integer multiple of *Hi*. In other words, "The number of blocks in width" in the table should be an integer multiple of (8x*H1*). Similarly, if picture height *Y* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of lines to complete bottom-most block-row. If the component is to be interleaved, the decoding process also needs to extend the number of lines by one or more additional block-rows so that the number of block-row is an integer multiple of (8x*V1*). For example, if non-interleaved YUV411 with *X*=270, then "The number of blocks in width" shall be (270 + 7) / 8 = 34, where "/" is integer division. Therefore, **FrameWidthInBlksMinus1** is set to 33. However, for interleaved data, "The number of blocks in width" shall be ((270 + 31) / 32) x 4 = 36. Therefore, **FrameWidthInBlksMinus1** is set to 35.

VertUpSamplingEnb is used to convert an input chroma420 to an output chroma422 in the surface format of YUY2 or UYVY. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV420, and OutputFormatYUV should be set to YUY2 or UYVY. Vertical 2:1 up-sampling is only applied to chroma blocks where each line of 8x8 block pixels is replicated to make 8x16 U/V blocks. For example:



VertDownSamplingEnb is used to convert an input chroma422 to an output chroma420 in the surface format NV12. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV422H_2Y or YUV422H_4Y, and OutputFormatYUV should be set to NV12. Combined with AvgDownSampling flag, the following table and figures show the down-sampling methods.

VertDownSamplingEnb	AvgDownSampling	Down-Sampling Methods
0	0 or 1	No down-sampling.
1	0	Drop every other line: 
1	1	Average vertically neighboring two pixels:

VertDownSamplingEnb	AvgDownSampling	Down-Sampling Methods
		

The recent history forJPEG Decoder Commands are described in the following:

-
- InputFormat is the same as IVB, and should be programmed the same way as IVB.
- If the InputFormat is YUV400 or YUV444 or YUV411, then output cannot be NV12, YUY2 or UYVY, it has to be planar (like legacy IVB). But for 420 and 422 InputFormat, there's a choice of having Planar, NV12, YUY2 or UYVY OutputFormat and the surface state should be programmed accordingly.
- Refer "Output Format YUV" field for more details.

MFX_JPEG_HUFF_TABLE_STATE

JPEG Encoder Commands

JPEG Encoder Command Sequence:

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_JPEG_PIC_STATE

MFX_FQM_STATE (One each for Luma, CB and CR)

MFC_JPEG_ HUFF_TABLE_STATE (Huffman table 0 and 1 need two commands to be issued).

MFC_JPEG_SCAN_OBJECT

MFX_PAK_INSERT_OBJECT (Multiple commands can be given based on the need)

Following are JPEG Encoder Commands:

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.

Programming Note: For completion of partial MCUs in JPEG encoding, it is important to note the following:

If the image's dimensions are not an exact multiple of the MCU size, the encoded data should include padding to round up to the next complete MCU, which is called completion of partial MCU. If the number of lines is not aligned with MCU structure (not a multiple of MCU size, i.e. 8, 16, 32), the encoding process needs to extend the number of lines to complete the bottom-most MCU-row. Similarly, if the number of samples per line is not aligned with MCU structure, the encoding process needs to extend the number of columns to complete the right-most sample MCUs. JPEG standard recommends that any incomplete MCUs be completed by replication of the right-most column and the bottom line of each component Y, U, and V.

The following equations are used to set the command for encoding partial MCUs.

FrameWidthInBlksMinus1 = (((*X* + (H_1 *8 -1)) / (H_1 *8)) * H_1) – 1

FrameHeightInBlksMinus1 = (((*Y* + (V_1*8 -1)) / (V_1*8)) * V_1) – 1

```
For YUV400,
    PixelsInHoriLastMCU = X % 8
    PixelsInVertLastMCU = Y % 8

For YUV420,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 16 if Y % 2 = 0, ((Y % 16) + 1) % 16 if Y % 2 = 1

For YUV422H_2Y,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 8

    X: the number of samples per line in Y-image

    Y: the number of lines in Y-image

    H1: horizontal sampling factor of Y-image in the Frame header

    V1:  vertical sampling factor of Y-image in the Frame header
```

Note that PixelsInHoriLastMCU=0 does not mean the num of pixels in the right-most MCUs = 0, but does mean that the right-most MCUs are fully filled with pixels, i.e., not a partial MCU.

For example, for input image dimension 17x26 pixels and an interleaved Scan, the following equations and the table show how to set the command for each OutputMcuStructure.

	YUV400	YUV420	YUV422H_2Y
MCU size of Y	8x8	16x16	16x8
MCU size of U and V	8x8	8x8	8x8
H_1 and V_1	1, 1	2, 2	2, 1

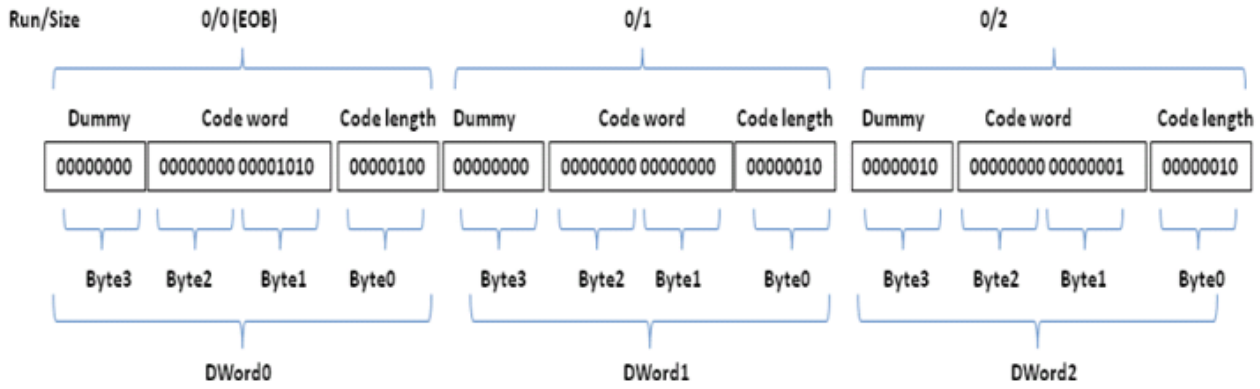
	YUV400	YUV420	YUV422H_2Y
FrameWidthInBlksMinus1	2	3	3
FrameHeightInBlksMinus1	3	3	3
PixelsInHoriLastMCU	1	2	2
PixelsInVertLastMCU	2	10	2

MFC_JPEG_SCAN_OBJECT
MFC_JPEG_ HUFF_TABLE_STATE

The JPEG standard Table K.5 shows the real table of code length and code word as follows:

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

It is not necessary to send Run/size in the command as driver will send the increasing order of run/size. Each symbol aligns to a DWord with the following byte structure. Each DWord (a group of 4 bytes) contains Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy.



Driver will program to always send 12 pairs of Code length and Code Word in DC coefficient table and 162 pairs in AC coefficient table. When a Huffman table contains valid full entries of Run/Size, all the Code word and Code length will not be zero. If a Huffman table is customized or optimized, the table can contain smaller set of Code length and Code Word, i.e., the number of entries of the real Huffman table will be less than 12 for DC, or less than 162 for AC. For the customized Huffman table, driver will set the missing entry (Run/Size) to Code length = 0 and Code word = 0.

MFX_PAK_INSERT_OBJECT

More Decoder and Encoder

MFD IT Mode Decode Commands

This is the decoder-only command to support the IT-mode specified in DXVA interface.

Only MPEG2-IT mode is supported.

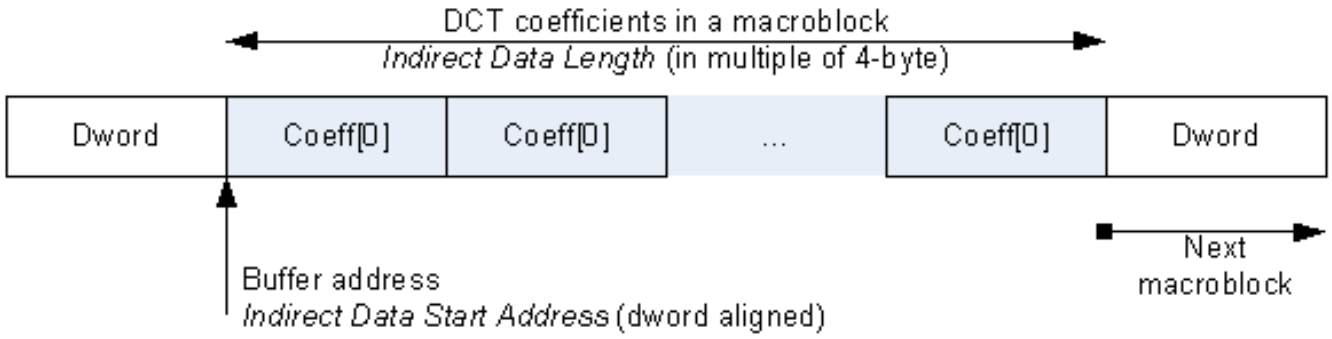
MFD_IT_OBJECT

Common Indirect IT-COEFF Data Structure

Transform-domain residual data block in MPEG2-IT mode follows the same data structure.

The indirect IT-COEFF data start address in MFD_IT_OBJECT command specifies the doubleword aligned address of the first non-zero DCT coefficient of the first block of the macroblock. Only the non-zero coefficients are present in the data buffer and they are packed in the 8x8 block sequence of Y0, Y1, Y2, Y3, Cb4 and Cr5, as shown in *Common Indirect IT-COEFF Data Structure*. When an 8x8 block is further subdivided into 4x4 subblocks, the coefficients, if present, are organized in the subblock order. The smallest subblock division is referred to as a **transform block**. The indirect IT-COEFF data length in the command includes all the non-zero coefficients for the macroblock. It must be doubleword aligned.

Structure of the IDCT Compressed Data Buffer



Each non-zero coefficient in the indirect data buffer is contained in a doubleword-size data structure consisting of the coefficient index, end of block (EOB) flag and the fixed-point coefficient value in 2's compliment form. As shown in *Common Indirect IT-COEFF Data Structure*, *index* is the row major 'raster' index of the coefficient **within a transform block** (*please note that it is not converted to 8x8 block basis*). A coefficient is a 16-bit value in 2's complement.

Structure of a transform-domain residue unit

DWord	Bit	Description
0	31:16	**Transform-Domain Residual (coefficient) Value.** This field contains the value of the non-zero transform-domain residual data in 2's compliment.
	15:7	Reserved: MBZ
	6:1	**Index.** This field specifies the raster-scan address (raw address) of the coefficient within the transform block. For a coefficient at Cartesian location (row, column) = (y, x) in a transform block of width W, Index is equal to (y * W + x). For example, coefficient at location (row, column) = (0, 0) in a 4x4 transform block has an index of 0; that at (2, 3) has an index of 2*4 + 3 = 11. The valid range of this field depends on the size of the transform block. Format = U6 Range = [0, 63]
	0	**EOB (End of Block).** This field indicates whether the transform-domain residue is the last one of the current transform block.

Allowed transform block dimensions per coding standard

Transform Block Dimension	MPEG2
8x8	Yes
8x4	No
4x8	No
4x4	No

Inline Data Description in MPEG2-IT Mode

The content in this command is similar to that in the MEDIA_OBJECT command in IS mode described in the Media Chapter.

Each MFD_IT_OBJECT command corresponds to the processing of one macroblock. Macroblock parameters are passed in as inline data and the non-zero DCT coefficient data for the macroblock is passed in as indirect data.

Inline Data Description in MPEG2-IT Mode depicts the inline data format. Inline data starts at dword 7 of MFD_IT_OBJECT command. There are 7 dwords total.

Inline data in MPEG2-IT Mode

DWord	Bit	Description
+0	31:28	**Motion Vertical Field Select.** A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the *ISO/IEC 13818-2* (see also §7.6.4).

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>28</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>29</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>30</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>31</td><td>1</td><td>1</td><td>3</td></tr>
</table>
Format = MC_MotionVerticalFieldSelect.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.
</td></tr>
<tr><td></td><td>27</td><td>Reserved (was Second Field)</td></tr>
<tr><td></td><td>26</td><td>Reserved. (HWMC mode)</td></tr>
<tr><td></td><td>25:24</td><td>
Motion Type. When combined with the destination picture type (field or frame) this Motion Type field indicates the type of motion to be applied to the macroblock. See ISO/IEC 13818-2 §6.3.17.1, Tables 6-17, 6-18. In particular, the device supports dual-prime motion prediction (11) in both frame and field picture type.
Format = MC_MotionType
<table>
<tr><th>Value</th><th>Destination = Frame
Picture_Structure = 11</th><th>Destination = Field
Picture_Structure != 11</th></tr>
<tr><td>‘00’</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>‘01’</td><td>Field</td><td>Field</td></tr>
<tr><td>‘10’</td><td>Frame</td><td>16x8</td></tr>
<tr><td>‘11’</td><td>Dual-Prime</td><td>Dual-Prime</td></tr>
</table>
</td></tr>
<tr><td></td><td>23:22</td><td>Reserved. (Scan method)</td></tr>
<tr><td></td><td>21</td><td>
DCT Type. This field specifies the DCT type of the current macroblock. The kernel should ignore this field when processing Cb/Cr data. See ISO/IEC 13818-2 §6.3.17.1. This field is zero if Coded Block Pattern is also zero (no coded blocks present).
0 = MC_FRAME_DCT (Macroblock is frame DCT coded).
1 = MC_FIELD_DCT (Macroblock is field DCT coded).
</td></tr>
<tr><td></td><td>20</td><td>Reserved (was Overlap Transform - H261 Loop Filter).</td></tr>
<tr><td></td><td>19</td><td>Reserved (was 4MV Mode - H263)</td></tr>
<tr><td></td><td>18</td><td>
Macroblock Motion Backward. This field specifies if the backward motion vector is active. See ISO/IEC 13818-2 Tables B-2 through B-4.
0 = No backward motion vector.
1 = Use backward motion vector(s).
</td></tr>
</table>

DWord	Bit	Description
	17	**Macroblock Motion Forward.** This field specifies if the forward motion vector is active. See *ISO/IEC 13818-2* Tables B-2 through B-4. 0 = No forward motion vector. 1 = Use forward motion vector(s).
	16	**Macroblock Intra Type.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used). See *ISO/IEC 13818-2* Tables B-2 through B-4. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**Reserved : MBZ**
	11:6	**Coded Block Pattern.** This field specifies whether blocks are present or not. Format = 6-bit mask. Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 Bit 6: Cr5
	5:4	Reserved. (Quantization Scale Code)
	3	LastMBInRow – This field indicates the last MB in each row.
	2:0	Reserved: MBZ
+1	31:16	**Reserved : MBZ**
	15:8	**VertOrigin - Vertical Origin** In unit of macroblocks relative to the current picture (frame or field).
	7:0	**HorzOrigin - Horizontal Origin** In unit of macroblocks.
+2	31:16	**Motion Vectors – Field 0, Forward, Vertical Component.** Each vector component is a 16-bit two's-complement value. The vector is relative to the current macroblock location. According to ISO/IEC 13818-2 Table 8, the valid range of each vector component is [-2048, +2047.5], implying a format of s11.1. However, it should be noted that motion vector values are sign extended to 16 bits.

DWord	Bit	Description
	15:0	**Motion Vectors – Field 0, Forward, Horizontal Component**
+3	31:16	**Motion Vectors – Field 0, Backward, Vertical Component**
	15:0	**Motion Vectors – Field 0, Backward, Horizontal Component**
+4	31:16	**Motion Vectors – Field 1, Forward, Vertical Component**
	15:0	**Motion Vectors – Field 1, Forward, Horizontal Component**
+5	31:16	**Motion Vectors – Field 1, Backward, Vertical Component**
	15:0	**Motion Vectors – Field 1, Backward, Horizontal Component**

Indirect Data Format in MPEG2-IT Mode

MPEG2-IT mode only contains IT-COEFF indirect data which is described in Section *Common Indirect IT-COEFF Data Structure*.

MFX Deblocking Commands

Following are MFX Deblocking Commands:

MFX_DBK_OBJECT

Encoder StreamOut Mode Data Structure Definition

When StreamOut is enabled, per MB (and/or per Slice, per Picture) intermediated coding data (for example, bit allocated for each MB, and so on) are sent to the memory in a fixed record format (and of fixed size) from the PAK. The per-MB records must be written in a strict raster order and with no gap (that is, every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

Adding per macroblock stream out for PAK is for the following purposes:

- Immediate multi-pass PAK (without host or EU intervention)
  - 3200-bit conformance
  - Re-quantization
- Providing information for host for offline processing
- Providing information for updated QP's

The description for the fixed format PAK streamout record:

Streamout Pointer: Use the existing streamout pointer and enabler

Per Macroblock Information (a fixed size structure)

DWord	Bit	Description
0	31:24	**MbQpY** - Actual QPY used by the macroblock.
	23:16	**MbClock16** – MB compute clocks in 16-clock unit.
	15:8	Reserved: MBZ
	7:4	Reserved: MBZ (future conformance flags)
	3	Reserved
	2	**MbRcFlag:** MB level Rate control flag(pass through) The same value as **RateControlCounterEnable** of MFX_AVC_SLICE_STATE Command
	1	**MbInterConfFlag:** MB level InterMB conformance flag to trigger mutli-pass 1- if total Bit Count of an inter macroblock is more than Inter Conformance Max size limit in the MFX_AVC_IMG_STATE Command
	0	**MbIntraConfFlag:** MB level IntraMB conformance flag to trigger mutli-pass 1- if total Bit Count of an intra macroblock is more than Intra Conformance Max size limit in the MFX_AVC_IMG_STATE Command
1	31:29	Reserved
	28:16	**MbBits:** Total Bit Count for the macroblock
	15:12	Reserved
	12:0	**MbHdrBits:** Header Bit count (bit count due to Pre-coefficient data) for the macroblock
2	31:27	Reserved
	26:0	**Cbp:** Coded Block Pattern of sub-blocks

DWord	Bit	Description
3	31:30	Reserved
	29	**IntraMBFlag**
	28:24	**MBType5Bits**
	23:17	Reserved
	16	**ClampFlag:** Coefficient clamping flag for RC (Status) 1 - Indicates if clamping of any coefficient is done on the macroblock for Rate Control
	15:0	Reserved (future QRC stat output)

PAK Frame Statistics StreamOut

The following frame statistics are written to memory at the conclusion of a frame. If Multipass occurs, these values are overwritten by the end of any subsequent passes of the current frame (hence it contains only the final pass statistics).

The streamout is done to the MB streamout surface, starting at the next CL boundary. If MB streamout is disabled, Frame level streamout starts with 0 offset.

MFX_PAK_FRAME_STATISTICS		
Source:	**VideoCS**	
Length Bias:	**2**	
DWord	**Bit**	**Description**
0	31:16	Reserved : MBZ
	15:0	SumSliceHeader – Report the total size (in bits) of all slice headers inserted into the bitstream for this frame.
1	31:0	SumMBHeader – Report the total size (in bits) of all MB headers (non coeff bits) inserted into the bitstream for this frame.
2	31:0	SumNZC – Report the total number of nonzero coefficients after quantization.
3	31:0	Reserved: MBZ
4	31:16	IntraMB16x16 – Count of # of MB's that were of type Intra 16x16
	15:0	IntraMB8x8 – Count of # of MB's that were of type Intra 8x8
5	31:16	IntraMB4x4 – Count of # of MB's that were of type Intra 4x4
	15:0	InterMB16x16 – Count of # of MB's that were of type Inter 16x16
6	31:16	InterMB16x8 – Count of # of MB's that were of type Inter 16x8
	15:0	InterMB8x16 – Count of # of MB's that were of type Inter 8x16
7	31:16	InterMB8x8 – Count of # of MB's that were of type Inter 8x8
	15:0	InterSkip16x16 – Count of # of MB's that were of type Inter 16x16 skip

8:49	31:0	RhoDomainStats – Each DW contains 1 of the 42 registers containing the raw Rho Domain coefficient metrics. DW 8 is QP 10 and DW 49 is QP51.
50	31:0	Reserved: MBZ

PAK Multi-Pass

Multi-Pass PAK Usages:

- Intra MB 3200-bit conformance
- Inter MB Re-quantization
- Frame level Re-quantization

How to Enable Multi-Pass PAK?

- Using the existing conditional batch buffer execution capability to skip/execute the second pass
    - How to dynamically change the condition?
        - Defined one error condition register with a mask. Do HW status page update at the end of the first pass. 0 means all OK, non-zero means there is an error condition, requiring second pass. Mask is used by the host to control what kind of multi-pass is intended.
        - For example, one error bit is 3200-bit conformance violation. Another error bit is the total bit count exceeds (too much or too little) the target range (need to define the target range in the state).
        - **The logic perfectly fits in the conditional batch buffer control logic that VCS has today in GT. There is no additional logic need to be added in VCS to support media functionality.** (**Batch Buffer Skip:** This field only takes effect if Compare Semaphore is set and the value at **Semaphore Address is NOT greater than the Semaphore Data Dword)**.
- Adding a picture level state command to enable and control the behavior of the second pass PAK
    - How to control the re-PAK? Added 3 conformance flags (error registers) in the per-MB streamout. Then the error control is based on the error register and the mask defined in picture level states. There are 8 register flags defined out of which only the 3200-bit case has usage model defined for today. The rest are left for future usage.

Issues and Limitations:

- There is no programmable engine in MFX for flexible control: Therefore, whatever we have defined must consider flexibility

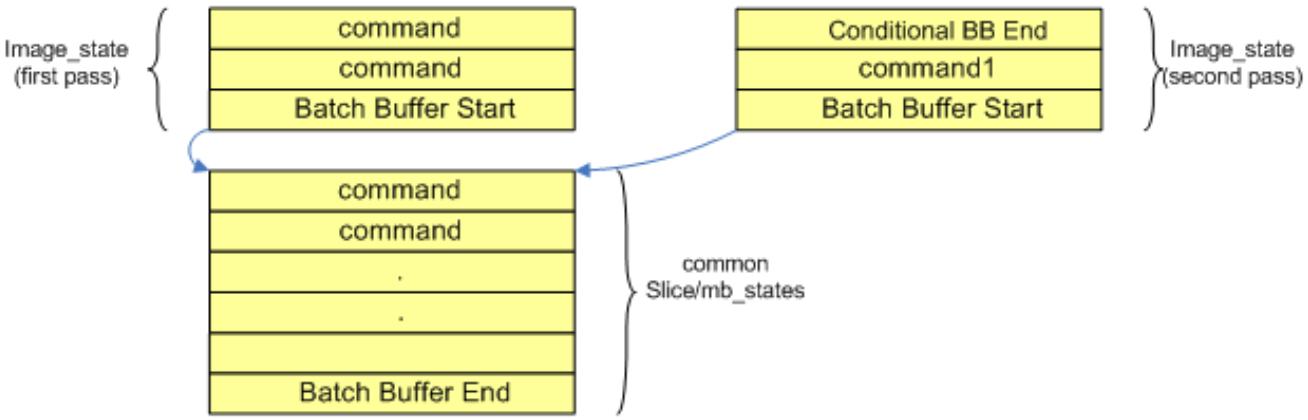
Following 2 MI packets are used inside VCS without any change to support Multipass-PAK behaviour.

- MI_Conditional_Batch_Buffer_End
- Memory Interface Registers

Driver Usage

Driver places Image states in one batch buffer and all slice level and macroblock level states into another batch buffer and link 2 batch buffers. Also replicate Image states with multipass changes in another batch buffer link them to slice/macroblock batch buffer. In this way, only Image states are replicated but not the slice/macroblock states. The image states includes all buffers defined at image(indirectMV, original pixel buffer, etc). Following changes are needed in the Multipass Image State,

- **Reset- Stream-Out Enable(disable stream out in the second pass)**
- **Set- MacroblockStatEnable (enable reading of macroblock status buffer)**
- **Reset- 3200-bit conformance (do not report 3200-bit conformance)**



Define Conditional Batch Buffer End for CS/VCSVINunit

Programming Reference

Monochrome Picture Processing

Monochrome picture is specified using the Surface State with Surface Format of 12. Therefore, MFX hardware, in either decode or encode mode, does not generate any read or write traffic for U/V components. The motivation for this bandwidth optimization is that monochrome video coding might be used for wireless display.

For Encoder:

1. No read in UV original components
2. Processing UV component - no
3. Reconstructed UV component reference picture - no
4. Filter UV component - no

For Decoder:

1. VLD mode: There is no color component coming out of the decoding pipeline in Monochrome mode and so no processing and not writing output.

2. IT mode: There is no color component in the coefficient buffer, and so no processing and not writing output.

Context Switch

There is no pre-emption for the BCS pipeline; hence every command buffer is required to contain all the states setup (preamble). Specifically, CPU can not interrupt the BCS-BSD pipe, to stop the operation in the middle of decoding a bitstream data.

Switch of contexts can only be performed at picture boundary.

No state needs to be saved.

PMSI Support

Pipeline Flush

Implicit flush for AVC and VC1 is performed at the end of Slice : for MPEG2 is done when a new image/picture command is issued. Because MPEG2 a slice can be one MB, no point to flush. MPEG2 will snoop the next command if it is an img_state command.

Explicit flush MI (1 bit to do media pipeline vs Gx pipeline) flush and cache flush (switch reference frame) – MI flush has bit to do cache flush. MI flush is for driver synchronization.

MMIO Interface

A set of registers are defined and accessible through MMIO interface to serve multiple purposes:

- Use for system configuration
- For accessing Performance counters

The following is the table for all the MMIO addresses for MFX.

Decoder Registers

Following are Decoder Registers:

MFD_ERROR_STATUS - MFD Error Status

AVC CAVLC

AVC CABAC

VC1

MPEG2

JPEG

MFD_PICTURE_PARAM - MFD Picture Parameter

MFX_STATUS_FLAGS - MFX Pipeline Status Flags

MFX_FRAME_PERFORMANCE_CT - MFX Frame Performance Count

MFX_SLICE_PERFORM_CT - MFX Slice Performance Count

MFX_MB_COUNT - MFX Frame Macroblock Count

MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count

MFX_LAT_CT1 - MFX_Memory_Latency_Count1

MFX_LAT_CT2 - MFX Memory Latency Count2

MFX_LAT_CT3 - MFX Memory Latency Count3

MFX_LAT_CT4 - MFX Memory Latency Count4

MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count

MFX_READ_CT - MFX Frame Motion Comp Read Count

MFX_MISS_CT - MFX Frame Motion Comp Miss Count

Encoder Registers

Following are the Encoder Registers:

Register
MFC_VIN_AVD_ERROR_CNTR - MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter .
MFC_BITSTREAM_BYTECOUNT_FRAME - Reported Bitstream Output Byte Count per Frame Register
MFC_BITSTREAM_SE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only Register
MFC_AVC_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register
AVC_CABAC_INSERTION_COUNT - MFC_AVC_CABAC_INSERTION_COUNT
MFC_AVC_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register
MFC_IMAGE_STATUS_MASK - MFC Image Status Mask
MFC_IMAGE_STATUS_CONTROL - MFC Image Status Control
MFC_QUP_CT - MFC QP Status Count
MFC_BITSTREAM_BYTECOUNT_SLICE - Bitstream Output Byte Count Per Slice Report Register
MFC_BITSTREAM_SE_BITCOUNT_SLICE - Bitstream Output Bit Count for the last Syntax Element Report Register
MFX_PAK_ERROR Register
MFX_PAK_WARNING Register
MFX_VP8_CNTRL_MASK - Reported BitRateControl parameter Mask
MFX_VP8_CNTRL_STATUS - Reported BitRateControl parameter Status
MFX_VP8_FRM_BYTE_CNT - Reported Final Bitstream Byte Count
MFX_VP8_FRM_ZERO_PAD - Reported Frame Zero Padding Byte Count
MFX_VP8_BRC_DQindex - Reported BitRateControl DeltaQindex
MFX_VP8_BRC_DLoopFilter - Reported BitRateControl DeltaLoopFilter
MFX_VP8_BRC_CumulativeDQindex01 - Reported BitRateControl CumulativeDeltaQindex and Qindex 01
MFX_VP8_BRC_CumulativeDQindex23 - Reported BitRateControl CumulativeDeltaQindex and Qindex 23
MFX_VP8_BRC_CumulativeDLoopFilter01 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 01
MFX_VP8_BRC_CumulativeDLoopFilter23 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 23
MFX_VP8_BRC_Convergence_Status - Reported BitRateControl Convergence Status

MMIO Interface

A set of registers are defined and accessible through MMIO interface to serve multiple purposes:

- Use as Status register for Bit Rate Control
- Use for Context Switch in Multipass

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	12900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	12904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	12908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1290B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	12910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	12914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW	12918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW	1291C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW	12920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW	12924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW	12928	Enc
MFX_VP8_DEBUG_CPBAC0_Bottom	CPBAC0 engine Bottom State	RO	1292C	Enc
MFX_VP8_DEBUG_CPBAC0_RangeCount	CPBAC0 engine Range/Count State	RO	12930	Enc
MFX_VP8_DEBUG_CPBAC0_Misc	CPBAC0 engine Misc. States	RO	12934	Enc
MFX_VP8_DEBUG_CPBAC1_Bottom	CPBAC1 engine Bottom State	RO	12938	Enc
MFX_VP8_DEBUG_CPBAC1_RangeCount	CPBAC1 engine Range/Count State	RO	1293C	Enc
MFX_VP8_DEBUG_CPBAC1_Misc	CPBAC1 engine Misc. States	RO	12940	Enc

The following registers are the same as above except they have a different Address Offset. They are used if the second VDbox (VP8 Encoder) exists.

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	1C900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	1C904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	1C908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1C90B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	1C910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	1C914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW	1C918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW	1C91C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW	1C920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW	1C924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW	1C928	Enc
MFX_VP8_DEBUG_CPBAC0_Bottom	CPBAC0 engine Bottom State	RO	1C92C	Enc
MFX_VP8_DEBUG_CPBAC0_RangeCount	CPBAC0 engine Range/Count State	RO	1C930	Enc
MFX_VP8_DEBUG_CPBAC0_Misc	CPBAC0 engine Misc. States	RO	1C934	Enc
MFX_VP8_DEBUG_CPBAC1_Bottom	CPBAC1 engine Bottom State	RO	1C938	Enc
MFX_VP8_DEBUG_CPBAC1_RangeCount	CPBAC1 engine Range/Count State	RO	1C93C	Enc
MFX_VP8_DEBUG_CPBAC1_Misc	CPBAC1 engine Misc. States	RO	1C940	Enc

Row Store Sizes and Allocations

	AVC	VC1	MPEG2	JPEG	IT	ENC	SEC ENC
vin_vmx_pixcoefind_ addr[31:6]	Bitstream	Bitstream	Bitstream	Bitstream	VDS COEF	Orig Pix	BSP data
vin_vmx_mvbsdrs_ addr[31:6]	VAD BSD		VMD RS		VDS MV	MPC MV	
vin_vmx_mpcildbmpr_ addr[31:6]	VAM MPR				VDS ILDB	MPC RS	
vin_vmx_dmv*_ addr[31:6]	VAM DMV	VCP DMV					
vin_vmx_bp_addr [31:0]		VCP BP					

MPEG2 VLD Decoding Mode :

use BSD Row Store only, and

MPEG2 IT Decoding Mode :

MPEG2 IT mode does not need row-store

JPEG VLD Decoding Mode : no row store is needed

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 9: Media Video Enhancement (VEBOX) Engine

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Media VEBOX Introduction 1

Denoise 1

Deinterlacer 2

VEBOX Output Statistics 2

Overall Surface Format 2

Statistics Offsets 4

LACE and ACE 4

Per Command Statistics 5

FMD Variances and GNE Statistics 5

Skin-Tone Data 6

Gamut Compression: Out of Range Pixels 6

Histograms 7

The histograms are in separate surfaces called the ACE/LACE and RGB Histogram. 7

Ace Histogram 7

STMM / Denoise 8

VEBOX State and Primitive Commands 9

VEBOX State 9

DN-DI State Table Contents 10

VEBOX_IECP_STATE 10

VEBOX Surface State 11

Surface Format Restrictions 11

VEB DI IECP Commands 12

Media VEBOX Introduction

The VEBOX is an independent pipe with a variety of image enhancement functions.

The following sections are contained in Media VEBOX:

Feature
Denoise
Deinterlacer
VEBOX Output Statistics
VEBOX State

Denoise

This section discusses the Denoise feature in the chipset.

- **Denoise Filter** - detects noise in the input image and filters the image with either a temporal filter or a spatial filter. The temporal filter is applied when low motion is detected.
- **Chroma Denoise Filter** - detects noise in the U and V planes separately and applies a temporal filter.
- **Block Noise Estimate (BNE)** - as part of the Global Noise Estimate (GNE) algorithm, BNE estimates the noise over each block of pixels in the input picture.
- **Global Noise Estimate (GNE)** - GNE is calculated at the end of the frame by combining all the BNEs. The final GNE value is used to control the denoise filter for the next input frame. Noise estimates are kept between frames and blended together.

Motion Detection and Noise History Update

This logic detects motion of the current block for the denoise filter. The detected motion is then combined with motion detected in the co-located block of the past frame, and stored in the denoise history table.

Block Noise Estimate (Part of Global Noise Estimate)

The BNE estimates the amount of noise in each rectangular region of the input picture. The BNE estimate is computed separately for each color component in the input picture. The estimates from BNE are summed together to generate the Global Noise Estimate (GNE) for the entire input picture.

Temporal Filter

Temporal denoise is applied to each pixel based on the noise strength measured from the input pictures. Each pair of co-located pixels in the current and previous input pictures is blended together to generated the output pixel.

Context Adaptive Spatial Filter

For each pixel in the local neighborhood, its luma value is compared to the center pixel to be filtered.

Each pixel in the neighborhood for which the difference is less than good_neighbor_th is marked as a "good neighbor". The filtered output pixel is then equal to average of “good neighbor” pixels.

Chroma Noise Detection

The operation of chroma noise detection module is similar to the luma noise detection module. The U & V channels are processed individually to generate a noise estimate for each of the 2 channels.

Chroma Noise Reduction Filter

A simple and effective temporal-domain chroma noise reduction filter is applied. The Noise History is updated based on the motion detection result as before. The Noise History value is used to control the temporal denoise filter.

Denoise Blend

The denoise blend combines the temporal and spatial denoise outputs.

Deinterlacer

The deinterlacer (DI) takes the top and bottom fields of each input frame and converts them into two output frames. This block also gathers statistics from a film mode detector at the end of the frame. If the film mode detector determines that the input is progressive, then the input fields are put together directly to construct the progressive output frame.

Features:

- **Deinterlacer** - DI starts by estimating how much motion is present across the input fields. Low motion scenes are reconstructed by temporal deinterlacer, while high motion scenes are reconstructed by spatial deinterlacer.
- **Film Mode Detection** - This detector determines if the input fields are created by sampling film content and converting it to interlaced video. If so, the deinterlacer is turned off in favor of reconstructing the progressive output frame directly from adjacent input fields in the Progressive Cadence Reconstruction mode. Various sum-of-absolute differences are computed per block.
- **Chroma Upsampling** - If the input is 4:2:0, then chroma data is doubled vertically to 4:2:2. Chroma data is then processed by chroma deinterlacer or progressive cadence reconstruction.

VEBOX Output Statistics

Overall Surface Format

Statistics are gathered on both per-block (16x4) basis and per-frame basis. There are 16 bytes of encoder statistics data per 16x4 block, plus a variety of per frame data which are stored in a linear surface. The 16 bytes of encoder statistics per block are output if either DN or DI are enabled and are organized into a surface with a pitch equal to the output surface width rounded to the next higher 64 boundary (so that each line starts and ends on a cache line boundary). The height of the surface is ¼ of the height of the output surface. If both DN and DI are disabled then the encoder stats are not output and the per frame information is output at the base address. The encoder statistics can be disabled with a state variable to lower the output bandwidth. The offsets for the other statistics do not change.

The per frame information is written twice per frame to allow for a 2 slice solution - in a single slice the second set of data will be all 0. The final per frame information is found by adding each individual Dword, clamping the data to prevent it from overflowing the Dword.

The Deinterlacer outputs two frames for each input frame. When IECP is applied to the Deinterlacer output, separate GCC and STD per frame statistics are created for each output frame. In this case, 4 copies of the per frame information are written - two copies for the two slice solution times two output frames. For the case of DN and no DI, only the first set of per frame statistics will be written.

Statistics Surface when DI Enabled and DN either On or Off

16 Bytes for X=0, Y=0	16 Bytes for X=16, Y=0	16 Bytes for X=32, Y=0	....	16 Bytes for X=Width-16, Y=0	
16 Bytes for X=0, Y=4					
16 Bytes for X=0, Y=8					
.....					
16 Bytes for X=0, Y=Height-4					
17 DWs Reserved		2DW of STD (Previous Slice 0)	2DW of GCC (Previous Slice 0)	11 DWs of Reserved	
11 DWs FMD (Slice 0)	6 DWs of GNE (Slice 0)	2DW of STD (Current Slice 0)	2DW of GCC (Current Slice 0)	2DWs of Auto focus (Current Slice 0)	9 DWs of Reserved
17 DWs Reserved		2DW of STD (Previous Slice 1)	2DW of GCC (Previous Slice 1)	11 DWs of Reserved	
11 DWs FMD (Slice 1)	6 DWs of GNE (Slice 1)	2DW of STD (Current Slice 1)	2DW of GCC (Current Slice 1)	2DWs of Auto focus (Current Slice 1)	9 DWs of Reserved

Statistics Surface when DN Enabled and DI Disabled

16 Bytes for X=0, Y=0	16 Bytes for X=16, Y=0	16 Bytes for X=32, Y=0	....	16 Bytes for X=Width-16, Y=0	
16 Bytes for X=0, Y=4					
16 Bytes for X=0, Y=8					
.....					
16 Bytes for X=0, Y=Height-4					
11 DWs White Balance Stats (Slice 0)	6 DWs of GNE (Slice 0)	2DW of STD (Slice 0)	2DW of GCC (Slice 0)	2DWs of Auto focus (Slice 0)	9 DWs of Reserved
11 DWs White Balance Stats (Slice 1)	6 DWs of GNE (Slice 1)	2DW of STD (Slice 1)	2DW of GCC (Slice 1)	2DWs of Auto focus (Slice 1)	9 DWs of Reserved

Statistics Surface when both DN and DI Disabled

11 DWs White Balance Stats (Slice 0)	6 DWs of GNE (Slice 0)	2DW of STD (Slice 0)	2DW of GCC (Slice 0)	2DWs of Auto focus (Slice 0)	9 DWs of Reserved
11 DWs White Balance Stats (Slice 1)	6 DWs of GNE (Slice 1)	2DW of STD (Slice 1)	2DW of GCC (Slice 1)	2DWs of Auto focus (Slice 1)	9 DWs of Reserved

When DN and DI are both disabled, only the per frame statistics are written to the output at the base address.

Statistics Offsets

The statistics have different offsets from the base address depending on what is enabled.

The encoder statistics size is based on the frame size:

```
Encoder_size = width * (height+3)/4
```

where

width is the width of the output surface rounded to the next higher 64 boundary and

height is the output surface height in pixels.

Offset	DI on	DI off + DN on	DI off + DN off
Per_Command_Previous_Slice0	Encoder_size	N/A	N/A
Per_Command_Current_Slice0	Encoder_size + 0x80	Encoder_size + 0x00	0x00
Per_Command_Previous_Slice1	Encoder_size + 0x100	N/A	N/A
Per_Command_Current_Slice1	Encoder_size + 0x180	Encoder_size + 0x80	0x80

LACE and ACE

The ACE Histograms are in a separate surface with the LACE Histograms and RGB Histograms.

For simplicity the offsets are the same in all modes - if a surface is disabled then it simply is not written, but the following surfaces offsets are not changed.

The starting offsets from the beginning of LACE/ACE/RGB Histogram Output are:

Offset	DI on	RGB on	DI on + RGB on
Red Histogram_Slice0	N/A	0x0000	0x0000
Green_Histogram_Slice0	N/A	0x0400	0x0400
Blue_Histogram_Slice0	N/A	0x0800	0x0800
Red Histogram_Slice1	N/A	0x0C00	0x0C00
Green_Histogram_Slice1	N/A	0x1000	0x1000
Blue_Histogram_Slice1	N/A	0x1400	0x1400
ACE_Histo_Previous_Slice0	0x1800	N/A	0x1800
ACE _Histo_Current_Slice0	0x1C00	0x1C00	0x1C00

Offset	DI on	RGB on	DI on + RGB on
ACE_Histo_Previous_Slice1	0x2000	N/A	0x2000
ACE_Histo_Current_Slice1	0x2400	0x2400	0x2400
LACE Histogram Current	0x1800	0x1800	0x1800
LACE Histogram Previous	0x1800+LACE_hist_size	N/A	0x1800+LACE_hist_size

Per Command Statistics

The Per Command Statistics are placed after the encoder statistics if either DN or DI is enabled. If the frame is split into multiple calls to the VEBOX, each call outputs only the statistics gathered during that call and software provides different base address per call and sums the resulting output to compute the per-frame data.

The final address of each statistic is:

> **Statistics Output Address** + **Per_Command_Offset** (pick the one for the slice desired and the current/previous frame for Deinterlacer) + **PerStatOffset**

FMD Variances and GNE Statistics

These are the 11 FMD variances (Variance 0 ~ 10) and Global Noise Estimate Statistics (Sums and Counts) collected in each VEBOX call.

Note that pixel values in blocks that are close to the edge of the frame (within a 16x4 block that intersects or touches the frame edge) are not used in the variance computation.

FMD variances are 0 when the Deinterlacer is disabled.

GNE estimates are 0 when the Denoise is disabled.

Counter Id	PerStatOffset	Associated Counter
0	0x00	FMD Variance 0
1	0x04	FMD Variance 1
2	0x08	FMD Variance 2
3	0x0C	FMD Variance 3
4	0x10	FMD Variance 4
5	0x14	FMD Variance 5
6	0x18	FMD Variance 6
7	0x1C	FMD Variance 7
8	0x20	FMD Variance 8
9	0x24	FMD Variance 9
10	0x28	FMD Variance 10
11	0x2C	GNE Sum Luma (Sum of BNEs for all passing blocks)
12	0x30	GNE Sum Chroma U

Counter Id	PerStatOffset	Associated Counter
13	0x34	GNE Sum Chroma V
14	0x38	GNE Count Luma (Count of number of block in GNE sum)
15	0x3C	GNE Count Chroma U
16	0x40	GNE Count Chroma V

Skin-Tone Data

The register Ymax stores the largest luma value. It is reset at the start of a command to zero.

The register Ymin stores the smallest luma value. It is reset at the start of a command to:

Reset Value
0xFFFF (65535 in 16 bits)

When comparing the luma values to Ymax or Ymin,

Feature
16-bit values are used: For 16bpc luma pixels: the entire 16 bits are used. For 8bpc luma pixels: 8 zeros are appended to the LSB.

There is also a 29-bit counter of all the skin pixels (Number of Skin Pixles).

Register values are 0 if the STD/STE function is disabled.

The registers are stored with the offsets as shown below.

PerStatOffset	Associated Register
0x044	Ymax (bits 31:16), Ymin (bits 15:0)
0x048	Number of Skin Pixels (bits [28:0], other bits zero)

Gamut Compression: Out of Range Pixels

The statistics gathered for Gamut Compression are:

1. Count of out-of-range pixels (29 bits) and
2. Sum of the distances from out-of-range pixels to the closest range boundaries (32 bits).

If the sum is greater than the maximum 32-bit value, then it is clamped to the maximum 0xFFFFFFFF.

Both values are reset to zero at the start of each command.

Both values are zero if the GCC function is disabled.

PerStatOffset	Associated Register
0x04C	Sum of distances of out-of-range pixels (clamped to 0xFFFFFFFF)
0x050	Number of out-of-range pixels (bits [28:0], other bits are zero)

Histograms

The histograms are in separate surfaces called the ACE/LACE and RGB Histogram.

Ace Histogram

The Ace Histogram counts the number of pixels at different luma values.

It has 256 bins, each of which is 24 bits.

Any count that exceeds 24 bits is clamped to the maximum value.

The data is stored on DWord boundaries with the upper 8 bits equal to zero.

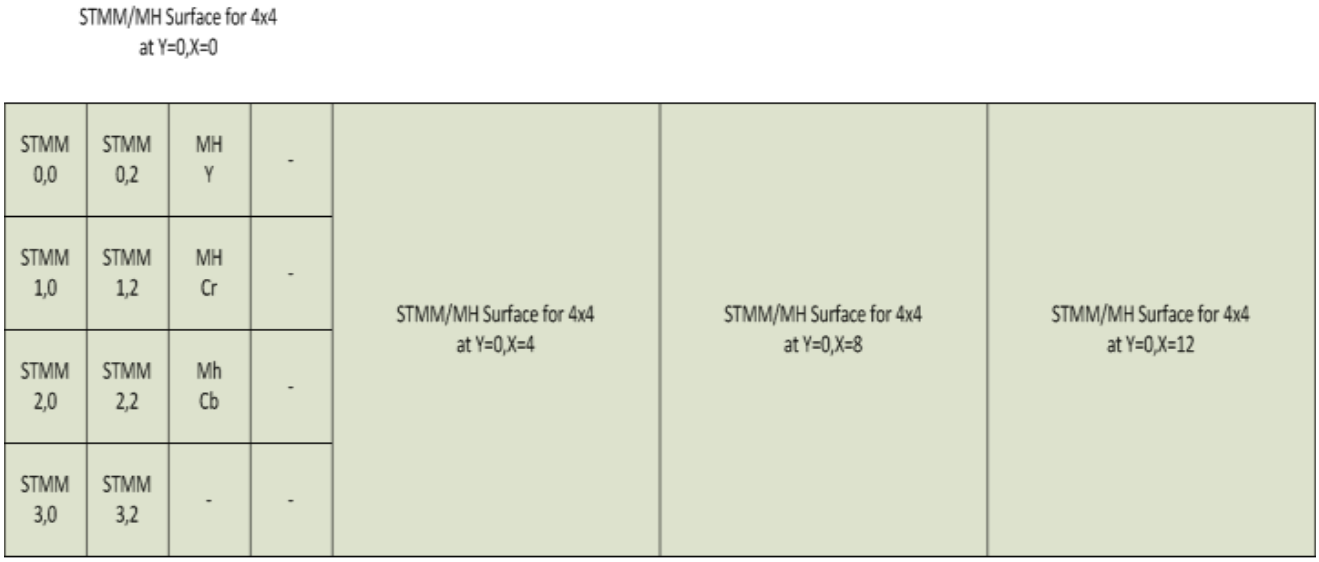
The Ace Histogram in non-capture mode is created by summing two independent 24-bit histograms, each of which is clamped to 24 bits. This means that the sum can reach 25 bits when both independent histograms are clamped or near clamped.

Y[9:2]	PerStatOffset	Associated Counter
0	0x000	ACE histogram, bin 0
1	0x004	ACE histogram, bin 1
2	0x008	ACE histogram, bin 2
...	...	...
255	0x3fc	ACE histogram, bin 255

STMM / Denoise

The STMM/Denoise history is a custom surface used for both input and output. The previous frame information is read in for the DN (Denoise History) and DI (STMM) algorithms; while the current frame information is written out for the next frame.

STMM / Denoise Motion History Cache Line



Byte	Data
0	STMM for 2 luma values at luma Y=0, X=0 to 1
1	STMM for 2 luma values at luma Y=0, X=2 to 3
2	Luma Denoise History for 4x4 at 0,0
3	Not Used
4-5	STMM for luma from X=4 to 7
6	Luma Denoise History for 4x4 at 0,4
7	Not Used
8-15	Repeat for 4x4s at 0,8 and 0,12
16	STMM for 2 luma values at luma Y=1,X=0 to 1
17	STMM for 2 luma values at luma Y=1, X=2 to 3
18	U Chroma Denoise History
19	Not Used
20-31	Repeat for 3 4x4s at 1,4, 1,8 and 1,12
32	STMM for 2 luma values at luma Y=2,X=0 to 1
33	STMM for 2 luma values at luma Y=2, X=2 to 3
34	V Chroma Denoise History
35	Not Used

Byte	Data
36-47	Repeat for 3 4x4s at 2,4, 2,8 and 2,12
48	STMM for 2 luma values at luma Y=3,X=0 to 1
49	STMM for 2 luma values at luma Y=3, X=2 to 3
50-51	Not Used
36-47	Repeat for 3 4x4s at 3,4, 3,8 and 3,12

VEBOX State and Primitive Commands

Every engine can have internal state that can be common and reused across the data entities it processes instead of reloading for every data entity.

There are two kinds of state information:

1. Surface state or state of the input and output data containers.
2. Engine state or the architectural state of the processing unit.

For example in the case of DN/DI, architectural state information such as denoise filter strength can be the same across frames. This section gives the details of both the surface state and engine state.

Each frame should have these commands, in this order:

1. VEBOX_State
2. VEBOX_Surface_state for input & output
3. VEB_DI_IECP

Alternatively VEBOX_Tiling_Convert can be used instead of VEB_DI_IECP.

VEBOX State

This chapter discusses various commands that control the internal functions of the VEBOX. The following commands are covered:

Feature
DN/DI State Table Contents
VEBOX_IECP_STATE
VEBOX_FORWARD_GAMMA_CORRECTION_STATE

Command
VEBOX_STATE
VEBOX_Ch_Dir_Filter_Coefficient

DN-DI State Table Contents

This section contains tables that describe the state commands that are used by the Denoise and Deinterlacer functions.

Command
VEBOX_DNDI_STATE

VEBOX_IECP_STATE

For all piecewise linear functions in the following table, the control points must be monotonically increasing (increasing continuously) from the lowest control point to the highest. Functions which have bias values associated with each control point have the additional restriction that any control points which have the same value must also have the same bias value. The piecewise linear functions include:

- For Skin Tone Detection:
    - Y_point_4 to Y_point_0
    - P3L to P0L
    - P3U to P0U
    - SATP3 to SATP1
    - HUEP3 to HUEP1
    - SATP3_DARK to SATP1_DARK
    - HUEP3_DARK to HUEP1_DARK
- For ACE:
    - Ymax, Y10 to Y1 and Ymin
    - There is no state variable to set the bias for Ymin and Ymax. The biases for these two points are equal to the control point values: B0 = Ymin and B11 = Ymax. That means that if control points adjacent to Ymin and Ymax have the same value as Ymin/Ymax then the biases must also be equal to the Ymin/Ymax control points based on the restriction mentioned above.
- Forward Gamma correction
- Gamut Expansion:
    - Gamma Correction

Inverse Gamma Correction

Command
VEBOX_IECP_STATE
VEBOX_STD_STE_STATE
VEBOX_ACE_LACE_STATE

Command
VEBOX_TCC_STATE
VEBOX_PROCAMP_STATE
VEBOX_CSC_STATE
VEBOX_ALPHA_AOI_STATE
VEBOX_FRONT_END_CSC_STATE

Command
VEBOX_VERTEX_TABLE
VEBOX_CAPTURE_PIPE_STATE
VEBOX_FORWARD_GAMMA_CORRECTION_STATE
VEBOX_RGB_TO_GAMMA_CORRECTION

VEBOX Surface State

Command
VEBOX_SURFACE_STATE

Surface Format Restrictions

The surface formats and tiling allowed are restricted, depending on which function is consuming or producing the surface.

Surface Formats - Feature Notes

Feature
Surfaces are 4 kb aligned, chroma X offset is cache line aligned (16 byte).
If Y8/Y16 is used as the input format, it must also be used for the output format (chroma is not created by VEBOX).
If IECP and either DN or DI are enabled at the same time, it is possible to select any input that is legal for DN/DI and any output which is legal for IECP. The only exception is that if DN or DI are enabled, the IECP is not able to output P216 and P016.
16-bit data from IECP or DN is rounded when converting to 8-bit output formats.
High Speed Bypass has the same format limitations as IECP Input/Output, but the surface formats for the input and output must be the same.
Capture Input is only linear Bayer Surface Format.
Input formats for Demosaic, White Balance, Vignette and Black Level Correction must be linear Bayer.
For capture pipe, we **can** support the combination of DN and P216 and P016. For capture pipe with a P016 output the U/V output is not an average of the 4 component pixels, but the U/V for pixel 4 (the lower right pixel of the 4).

VEB DI IECP Commands

The VEB_DI_IECP command causes the VEBOX to start processing the frames specified by VEB_SURFACE_STATE using the parameters specified by VEB_DI_STATE and VEB_IECP_STATE.

Command
VEB_DI_IECP Command
VEB_DI_IECP Command
VEB_TilingConvert Command

The Surface Control bits for each surface:

Command
VEB_DI_IECP
VEB_DI_IECP Command Surface Control Bits
VEBOX_TILING_CONVERT

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 10: HEVC Codec Pipeline (HCP)

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

High Efficiency Video Coding (HEVC) Introduction 1
Scope 1
Summary of Features 1
HCP Hardware Pipeline Features 1
HEVC Decoder Features 2
HEVC Encoder Features 2
Architecture Overview 5
VP9 Decoder 5
HCP Command Summary 5
Workload Command Model 5
HCP Command Sequence Examples ..... **Error! Bookmark not defined.**
HCP Decoder Command Sequence 7
HCP Encoder Command Sequence 8
Memory Address Attributes 10
HCP Pipe Common Commands 10
Buffer Size Requirements 11
VP9 Common Commands 13
HCP Common Commands 13
VP9 Commands 14
Tile Size and CU Stream-out Records 14
Stream-in Probability Table 1
Stream-in formats for creating compressed header 1
Probability Statistics Counters Table 2
VP9 PAK Quant Lookup Tables 1
SB, CU/PU and TU Sizes – Encoder Only 15
Definition of the VP9 CU Record Structure - Encoder Only 16
Allowed SB Size Encoder Only 21
HCP Commands 21
Multipass flow during BRC and SAO 22
CU and Slice level stat streamOut 23
Definition of the CU Record Structure for Ext Interface – Encoder Only 24

LCU, CU, TU, and PU Sizes – Encoder Only 28
Allowed LCU Size – Encoder Only 32
HEVC Error Concealment 33
VP9 Register Definitions 35
Register Attributes Description 35
VP9 Register Map 35
VP9 Encoder Register Descriptions 35
HEVC Register Definitions 35
Register Attributes Description 36
HCP Decoder Register Map 36
HCP Decoder Register Descriptions 36
HCP Encoder Register Map 36
HCP Encoder Register Descriptions 36
HUC Status Register Descriptions 37
Acronyms and Applicable Standards 37
Acronyms and Abbreviations 37

High Efficiency Video Coding (HEVC) Introduction

The HEVC Codec Pipeline (HCP) is a fixed function hardware video codec responsible for decoding and encoding HEVC (High Efficiency Video Coding) video streams.

Scope

The primary scope of the HCP BSpec document is to provide a description of the HCP commands processed by the Video Command Streamer (VCS). The secondary scope is to provide a description of the status registers on the Message Channel Interface to support encoding and decoding of the HEVC video format.

The BSpec sections include:

- Summary of Features
- Architecture Overview
- Commands
- Register Definitions
- Acronyms and Applicable Standards

Summary of Features

The following sections define the HEVC Decoder and Encoder general features and the features specific to HEVC decoding and encoding, respectively.

HCP Hardware Pipeline Features

- Supports both decoder and encoder functions, setup on a per picture basis:
  - Hardware acceleration provides Ctb/CU level decode and encode.
  - No context switch is supported within a frame process.
- Supports Video Command Streamer (VCS):
  - Shared with MFX HW pipeline, and at any one time, only one pipeline (MFX or HCP) and one operation (decoding or encoding) can be active.
- Supports Message Channel Interface:

Feature
Supports Tile-YS and Tile-YF.
Supports Tile-Y Legacy.

- Supports NV12 video buffer plane:
  - Supports 4:2:0, 8-bit per pixel component (Y, Cb and Cr) video.
- Supports 8Kx8K frame size.

HEVC Decoder Features

- Supports full-featured HEVC Main Profile standard, up to Level 6.2.
- Supports the long format HW decoding interface:
  - All headers (SPS, PPS, Slice Header) are parsed and decoded outside the HCP HW pipeline. They are then fed to the HW through a set of HCP state commands.
- Supports inner-loop decode with hardware entry points for Encoder.
- Error detection/resiliency down to the Ctb/CU level.

HEVC Encoder Features

- Supports ENC-PAK architecture
- Supports multiple pass BRC rate control operation flow
- Supports the HEVC Main Profile standard, with certain restrictions on the feature set and coding parameters, listed in the following table:

HEVC Encoder Features and Restrictions

Note that there is a difference between what PAK supported and what ENC supported. A feature/function that is supported in the PAK, does not necessary being supported by ENC and MediaSDK and the like.

Coding Tool	Gen9 Support	Gen9 Restriction	Comments
LCU Size	Yes (spec)		Support all 3 sizes: 16x16, 32x32, 64x64.
CU Size	Yes (spec)		Support 8x8, 16x16, 32x32, 64x64. Max 64 CU per LCU; min. CU size intel supported is 8x8 for all LCU size.
PU Partition	Yes (spec)		Support all inter symmetric (square) and asymmetric (non-square) PU partitioning, according to HEVC spec. PU Size for inter : Smallest allowed is 4x8 and 8x4, and they cannot be bidirectional. Inter 4x4 PU is not allowed in Main Profile.
TU QuadTree		Partial (intel) max depth is set to 3	Max depth is 3 (64x64 CU with 4x4 TU). Decoder supports this depth, but probably no need to search this for encode. Better to just split CU. HM common conditions set the max to 2 for both inter and intra. Intel Encoder only supports 2 levels of quad-tree. That is, max_transform_hierarchy_depth_inter/intra <= 2. Max num of TUs per CU is 16.
AMP	Yes (spec)		Asymmetric Motion Partition (rectangular PU partitioning - 2NxnU, 2NxnD, nLx2N or nRx2N). Available only for 64x64 to 16x16 CU.
AMVP	Yes (spec)		Adaptive/Advanced Motion Vector Prediction : spatial and PU-based temporal co-located MV candidates with scaling. Logic available from decoder. HW PAK is supporting temporal MV candidates.

Coding Tool	Gen9 Support	Gen9 Restriction	Comments
Merge	Yes (spec)		Merge Skip and Regular Merge. Max. 5 MV Merge candidates (4 spatial + 1 temporal co-located) with scaling. Logic available from decoder. [merge_flag, merge_index, skip_flag]
Parallel Motion Merge		No (intel)	Tool for parallel decode of MVs. Since this isn't constrained by Main Profile, the decoder has to meet performance targets in the worst case anyway.
MC Interpolation Filter	Yes (spec)		¼-pel Luma MV precision, 1/8-pel Chroma MV precision. 8-tap Luma filtering for both ½-pel and ¼-pel locations (1-pass). 4-tap Chroma filtering. Use separable (first horizontal then vertical 1-D filtering) filter coefficients. Not all filter kernels are symmetrical and can map into simple arithmetic. It is a DCT-IF based filter. All operations are within 16-bit data.
Weighted Prediction	Yes (spec)		Free for PAK since decoder already has it. Cross fade and fade-in/out detection required by VME. VME must supply weights, implicit weighted prediction is not supported in HEVC. Only explicit weight is supported. They are both unidirectional and bidirectional.
Combined Reference Frame List		No (intel)	Combine List0 and List1 into a single list to remove uni-prediction signaling overhead.
Intra modes	Yes (spec)		33 directions and DC/Planar modes, with adaptive pre-filtering on reference pixels and boundary smoothing. Intel ENC-VME Mode search reduction for large CUs (64x64 and 32x32) for performance speed up with minimal loss.
IPCM (intra)		No (intel)	Can be disabled completely by SPS, or only allowed at certain CU sizes. No bit maximum on CUs in any profile/level (yet), so as of today there's no mandate to support this for encode. Gen9 PAK does not support IPCM, so ENC cannot generate IPCM mode.
Constrained Intra		No (intel)	Allow only intra neighboring blocks for current block intra-prediction. Enabling this is a coding loss and does not result in a performance improvement in HW designs.
2D DCT Transform	Yes (spec)		Square shape only; 32x32, 16x16, 8x8 and 4x4.
Transform Skip Evaluation		No (intel) ENC will estimate the use of transform skip	Significant coding gains for screen content (PowerPoint etc.). This tool is disabled in the common conditions but isn't explicitly disallowed by Main Profile. FQ is not bypass.
Sign bit hiding		No (intel)	Coding gain by removing one bypass bin per TU. Requires some smarts in the PAK.
Trellis		No (intel)	Trellis Quantization
SAO		No (intel)	Difficult to implement in single pass, performance impact in 2-pass or with previous frame search. Needs investigation. Decoder will support it.

Coding Tool	Gen9 Support	Gen9 Restriction	Comments
Loop Filter across tiles/slices boundary		No (intel)	Can be disabled for tiles and or slices in SPS, so that filter across all tiles and slices boundaries. Main profile doesn't constrain.
Scaling List	Yes (spec)		This uses the default (or custom) qp adjustment on a per-frequency basis within a TU. Good coding improvement over flat scaling.
dQP		Partial (intel) Yes for LCU; No for CU	Being able to change QP per LCU or even up to once per 8x8 CU can lead to significant coding gains. Not sure how easy it is for VME to decide qp values though.
Chroma QP offset		No (intel)	No ROI.
Dependent slices		No (intel)	It is now part of Main Profile, but Intel will not support it. SW can perform the slice repackaging without re-encoding.
Tiles		No (intel)	Although in Main Profile, it results in coding loss and doesn't improve performance on HW. SW parallel processing (multithreaded) tool
Wavefront (aka WPP)		No (intel)	Latest Main Profile spec has included Wavefront. We got a feedback that this feature is highly desirable to support high performance multithreading HEVC decoder.
Lossless coding		No (intel)	Note: this is not the same as IPCM. Also details of this are in flux.
Interlaced Video		No (intel)	Only progressive video encoding is supported.
LM mode	No (spec)		Chroma-from-luma intra prediction (Linear Mode) is not allowed in Main Profile (yet).
NSQT	No (spec)		Non square transform is not allowed in the Main Profile
ALF	No (spec)		Expensive and not in Main Profile currently. If decoder is going to support it, may consider for encoder as well.
Entropy slices	No (spec)		Not allowed in Main Profile, it is a SW parallel processing (multithreaded) tool.
Slice granularity != 0	No (spec)		Not allowed by Main Profile, and highly likely to be removed from the standard completely.

Architecture Overview

This section discusses HEVC features as follows:

Name
VP9 Decoder

VP9 Decoder

The HCP Decoder, following diagram, is a hardware based pipeline that accelerates the stream decode for an external Host Processor. The Host Processor is defined as a system processor external to the HCP that is responsible for high level parsing of the elementary stream and workload creation.

HCP Command Summary

The HCP is configured for encoding or decoding through a set of batch commands defined in the following sections. The software driver builds a frame level workload using these commands and stores these workloads in graphics memory where they are fetched by the Video Command Streamer (VCS) and presented to the HCP for processing. The commands are processed by the Workload Parser within the HCP and the hardware is configured by the Workload Parser prior to each frame level encode or decode. A workload is defined as a set of commands necessary to encode or decode one frame.

The software driver is required to read the HCP disable fuse to determine if the HCP is enabled. If it is disabled, then the software driver must not enable HCP batch commands to be sent to the HCP or a hang event may occur. Only when the HCP is enabled through the fuse, should the batch commands be sent to the HCP.

Workload Command Model

DWord0 of each command is defined in HCP DWord0 Command Definition. The HCP is selected with the **Media Instruction Opcode "7h**" for all HCP Commands.

HCP DWord0 Command Definition

DWord	Bits	Description
0	31:29	**Command Type** = PARALLEL_VIDEO_PIPE = 3h
	28:27	**Pipeline Type** = 2h
	26:23	**Media Instruction Opcode = Codec/Engine Name** = HCP = 7h
	22:16	**Media Instruction Command** = *<see Workload Command Model >*
	15:12	**Reserved: MBZ**

DWord	Bits	Description
	11:0	**Dword Length** (Excludes Dwords 0, 1) = <*command length*>

Each HCP command has assigned a media instruction command as defined in HCP Media Instruction Commands (Opcode=7h).

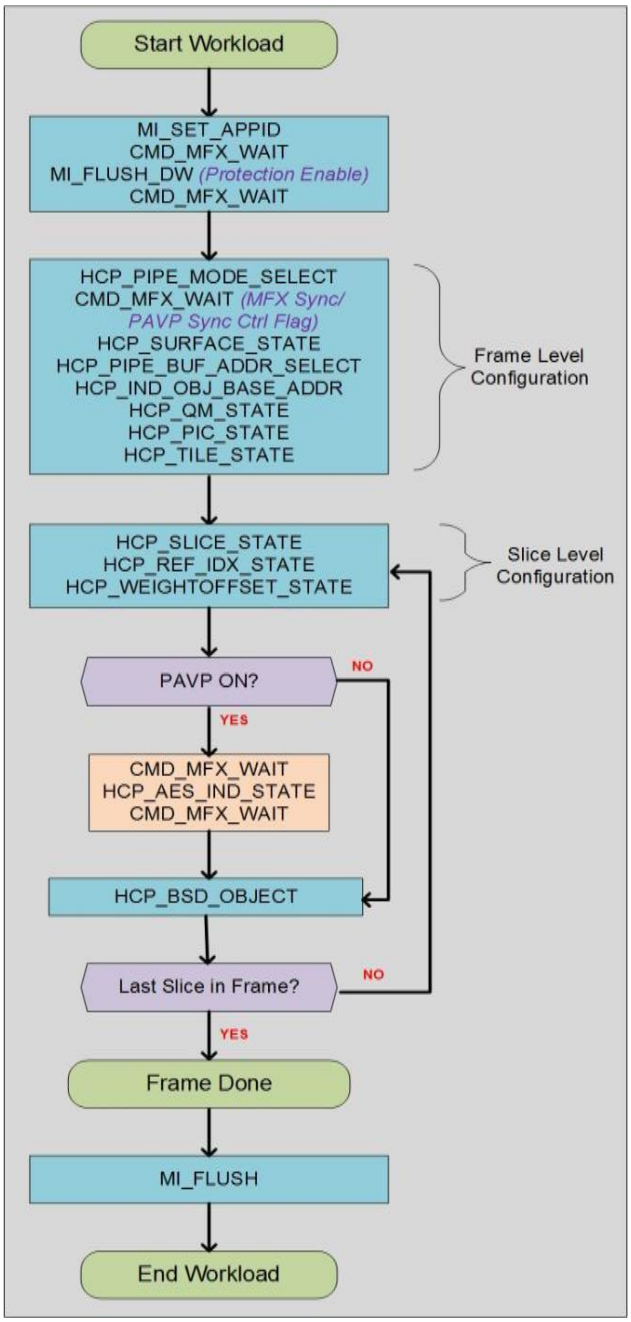
HCP Media Instruction Commands (Opcode=7h)

Media Instruction Command	Command DWord0 [22:16]	Gen9 Mode	Scope
HCP_PIPE_MODE_SELECT	0h	Enc/Dec	Picture
HCP_SURFACE_STATE	1h	Enc/Dec	Picture
HCP_PIPE_BUF_ADDR_STATE	2h	Enc/Dec	Picture
HCP_IND_OBJ_BASE_ADDR_STATE	3h	Enc/Dec	Picture
HCP_QM_STATE	4h	Enc/Dec	Picture
HCP_FQM_STATE (encoder only)	5h	Enc	Picture
Reserved	8h-Fh		
HCP_PIC_STATE	10h	Enc/Dec	Picture
HCP_TILE_STATE	11h	Dec	Picture
HCP_REF_IDX_STATE	12h	Enc/Dec	Slice
HCP_WEIGHTOFFSET	13h	Enc/Dec	Slice
HCP_SLICE_STATE	14h	Enc/Dec	Slice
Reserved	15h-1Fh		
HCP_BSD_OBJECT_STATE (decoder only)	20h	Dec	Slice
HCP_PAK_OBJECT (encoder only)	21h	Enc	LCU
HCP_INSERT_PAK_OBJECT (encoder only)	22h	Enc	Bitstream

HCP Decoder Command Sequence

The long format workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.

HCP Long Format Decode Workload Flowchart



HCP Encoder Command Sequence

For a single frame encoding process (w/o multiple slices per frame), the command sequence is listed below. There are no states saved between frame encoded in the HCP. There should be no other commands or context switch within a group of PAK OBJECT Commands, representing a complete slice. HCP and MFX share the same VCS, but there is no common encoding and decoding command that can be executed in both pipes mi_flush and MMIO commands.

------ Per Frame Level Commands

HCP_PIPE_MODE_SELECT

HCP_SURFACE_STATE

HCP_PIPE_BUF_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_FQM_STATE – issue n number of times

HCP_QM_STATE – issue n number of times

HCP_PIC_STATE

------- Per Slice Level Commands (2 cases)

------- A Frame with only 1 Slice:

HCP_REF_IDX_STATE – set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE – set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command – set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT – if header present at 1st slice start

------- A group of LCUs Per Slice

HCP_PAK_OBJECT

…

HCP_PAK_INSERT_OBJECT – if tail present at frame end

MI_FLUSH – when the frame is done

------ A Frame with Multiple Slices:

HCP_REF_IDX_STATE – set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE – set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command – set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT – if header present at 1st slice start of a frame

HCP_PAK_OBJECT - a group of LCUs for a slice or a frame

…

HCP_PAK_INSERT_OBJECT – if tail present at slice or frame end

HCP_REF_IDX_STATE – set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE – set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command – set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT – if header present at slice start

HCP_PAK_OBJECT - a group of LCUs for a slice or a frame

…

HCP_PAK_INSERT_OBJECT – if tail present at last slice end (frame end)

MI_FLUSH – when the frame is done

MFX_STITCH_OBJECT – a generic bitstream stitching command from MFX pipe

MI_FLUSH

MI_FLUSH is not allowed between Slices. HEVC CABAC has simplified its operation from AVC. There is no longer a BSP_BUF_BASE_ADDR_STATE Command, as only a small local internal buffer is needed for BSP/BSE row store. THE HCP PAK_INSERT_OBJECT has been designed to support both inline and indirectly payload. Nevertheless, the MFX_STITCH_OBJECT command can still be used to stitch HEVC bitstreams together, and is run in the MFX pipe. No HEVC specific STITCH command is implemented. The SURFACE_STATE command for HEVC is redesigned and much simplified from that of MFX pipe.

Memory Address Attributes

This section defines the memory address attributes for the third DWord of the HCP command buffer address.

NOTE: The first DWord defines the lower address range and the second Dword defines the upper address range in the HCP command buffer address.

MemoryAddressAttributes

HCP Pipe Common Commands

The HCP Pipe Common Commands specify the HEVC Decoder pipeline level configuration.

Commands
HCP_PIPE_MODE_SELECT (VideoCS)
HCP_SURFACE_STATE (VideoCS)
HCP_PIPE_BUF_ADDR_STATE (VideoCS)

Buffer Name	Minimum Size in CLs	Notes
Deblocking Filter Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Column Buffer	((picture_height_in_pixels + 6*pic_height_in_ctb + 31) & (-32)) » 3	Eq. ensures multiple of 4
Metadata Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu*8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu *16 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (all intra slices)	(picture_height_in_pixels + picture_height_in_pixels *8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Line Buffer (some inter slices)	(((picture_width_in_pixels+15)»4)*188 + pic_width_in_lcu*9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (some inter slices)	(((picture_width_in_pixels +15)»4)*172 + pic_width_in_lcu *9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (some inter slices)	(((picture_height_in_pixels +15)»4)*176 + picture_height_in_lcu *89 + 1023) » 9	Eq. ensures multiple of 2
SAO Line Buffer	(((picture_width_in_pixels » 1) + pic_width_in_ctb * 3)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Line Buffer	(((picture_width_in_pixels » 1) + picture_width_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Column Buffer	(((picture_height_in_pixels » 1) + pic_height_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2

Buffer Name	Minimum Size in CLs	Notes
Current and Collocated Motion Vector Temporal Buffer (lcu=16x16)	(((picture_width_in_pixels+63)»6)*((picture_height_in_pixels+15)»4)	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu>16x16)	(((picture_width_in_pixels+31)»5)*((picture_height_in_pixels+31)»5)	Eq. ensures multiple of 2

Internal Media Rowstore Table

If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

HEVC Pipeline	Frame Width	DAT	DF	SAO	DAT Addr	DF Addr	SAO Addr
HEVC	<= 2048	Y	Y	Y	0	64	320
	2048 < x <= 4096	Y	Y	N	0	128	N/A
	4096 < x <=8196	Y	N	N	0	N/A	N/A

Commands
HCP_IND_OBJ_BASE_ADDR_STATE (VideoCS)
HCP_QM_STATE (VideoCS)
HCP_FQM_STATE VideoCS

Buffer Size Requirements

HEVC Buffer Size Requirements (8 bit)

Buffer Name	Minimum Size in CLs	Notes
Deblocking Filter Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Column Buffer	((picture_height_in_pixels + 6*pic_height_in_ctb + 31) & (-32)) » 3	Eq. ensures multiple of 4
Metadata Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu*8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu *8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (all intra slices)	(picture_height_in_pixels + picture_height_in_lcu * 16 + 1023) » 9	Eq. ensures multiple of 2
Metadata Line Buffer (some inter slices)	(((picture_width_in_pixels+15)»4)*188 + pic_width_in_lcu * 9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (some inter slices)	(((picture_width_in_pixels +15)»4)*172 + pic_width_in_lcu * 9 + 1023) » 9	Eq. ensures multiple of 2

Buffer Name	Minimum Size in CLs	Notes
Metadata Tile Column Buffer (some inter slices)	**Equation:** (((picture_height_in_pixels +15)»4)*176 + picture_height_in_lcu * 89 + 1023) » 9	Eq. ensures multiple of 2
SAO Line Buffer	(((picture_width_in_pixels » 1) + pic_width_in_ctb * 3)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Line Buffer	(((picture_width_in_pixels » 1) + picture_width_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Column Buffer	(((picture_height_in_pixels » 1) + pic_height_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu=16x16)	(((picture_width_in_pixels+63)»6)*((picture_height_in_pixels+15)»4)	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu>16x16)	(((picture_width_in_pixels+31)»5)*((picture_height_in_pixels+31)»5)	Eq. ensures multiple of 2
SSE Line Buffer	(Picture_width_in_lcu + 2) « 4	

HEVC 10 bit Buffer Size Requirements

Buffer Name	Minimum Size in CLs	Notes
Deblocking Filter Line Buffer	((picture_width_in_pixels + 31) & (-32)) » **2**	Eq. ensures multiple of 4
Deblocking Filter Tile Line Buffer	((picture_width_in_pixels + 31) & (-32)) » **2**	Eq. ensures multiple of 4
Deblocking Filter Tile Column Buffer	((picture_height_in_pixels + 6*pic_height_in_ctb + 31) & (-32)) » **2**	Eq. ensures multiple of 4
Metadata Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu*8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu *8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (all intra slices)	(picture_height_in_pixels + picture_height_in_lcu * 16 + 1023) » 9	Eq. ensures multiple of 2
Metadata Line Buffer (some inter slices)	(((picture_width_in_pixels+15)»4)*188 + pic_width_in_lcu*9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (some inter slices)	(((picture_width_in_pixels +15)»4)*172 + pic_width_in_lcu *9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (some inter slices)	(((picture_height_in_pixels +15)»4)*176 + picture_height_in_lcu * 89 + 1023) » 9	Eq. ensures multiple of 2
SAO Line Buffer	(((picture_width_in_pixels » 1) + pic_width_in_ctb * 3)+15) & (-16)) » **2**	Eq. ensures multiple of 2

Buffer Name	Minimum Size in CLs	Notes
SAO Tile Line Buffer	(((picture_width_in_pixels » 1) + picture_width_in_ctb * 6)+15) & (-16)) » **2**	Eq. ensures multiple of 2
SAO Tile Column Buffer	(((picture_height_in_pixels » 1) + pic_height_in_ctb * 6)+15) & (-16)) » **2**	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu=16x16)	(((picture_width_in_pixels+63)»6)*((picture_height_in_pixels+15)»4)	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu>16x16)	(((picture_width_in_pixels+31)»5)*((picture_height_in_pixels+31)»5)	Eq. ensures multiple of 2
SSE Line Buffer	(Picture_width_in_lcu + 2) « 4	

Internal Media Rowstore Table

If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

HEVC Pipeline	Frame Width	DAT	DF	SAO	DAT Addr	DF Addr	SAO Addr
HEVC	<= 2048	Y	Y	Y	0	64	320
	2048 < x <= 4096	Y	Y	N	0	128	N/A
	4096 < x <=8196	Y	N	N	0	N/A	N/A

VP9 Common Commands

Commands
HCP_PIPE_MODE_SELECT
HCP_SURFACE_STATE
HCP_PIPE_BUF_ADDR_STATE
HCP_IND_OBJ_BASE_ADDR_STATE
HCP_VP9_SEGMENT_STATE
HCP_VP9_PIC_STATE

HCP Common Commands

HCP_PIC_STATE (VideoCS)

HCP_TILE_STATE (VideoCS)

HCP_REF_IDX_STATE (VideoCS)

HCP_WEIGHTOFFSET_STATE (VideoCS)

HCP_SLICE_STATE (VideoCS)

HEVC_VP9_RDOQ_STATE

VP9 Commands

HCP_BSD_OBJECT (triggers HW start)

HCP_VP9_PAK_OBJECT

Tile Size and CU Stream-out Records

CU statistics record (individual PUs per record down to 8x8 only) (Note: For Advanced BRC but not supported in HW yet)

Fields	Bits	
Skip	3:0	Indicates Skip flag Group 4 4x4s -> 4 bits
InterMode	11:4	InterMode: 0 NEARESTMV, 1 NEARMV, 2 ZEROMV, 3NEWMV Group 4 4x4s total 8 bits
Reserved	15:12	
NZ coeff count	28:16	Number of non-zero coeffs; sum of YUV, 13bits
Reserved	31:29	
NumBitsforCoeffs	47:32	Number of Bits for coefficients per block, 16bits
NumBitsforBlock	63:48	Number of Bits in block

Stream-in Probability Table

In Encoder mode, two sets of this table will be streamed out: one for the current frame probability update and one for future frame.

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
CL aligned	0	1	tx_probs_8x8 [0] [0..0]	100	100	0	0	0	0	MODE COUNTERS (counts tx)	0-17								
	1	1	tx_probs_8x8 [1] [0..0]	66	66	0	1	1											
	2	2	tx_probs_16x16 [0] [0..1]	20, 152	20, 152	0	2	2											
	4	2	tx_probs_16x16 [1] [0..1]	15, 101	15, 101	0	4	4											
	6	3	tx_probs_32x32 [0] [0..2]	3, 136, 37	3, 136, 37	0	6	6											
	9	3	tx_probs_32x32 [1] [0..2]	5, 52, 13	5, 52, 13	0	9	9											
	12	52	DUMMY	0, 0, 0, 0	0, 0, 0, 0	52	12												
CL aligned	64	3	coef_probs_4x4 [0] [0] [0] [0]	195, 29, 183	195,	0	12		8	COEFF COUNTERS (coeff_count_mod		0-28							

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		29, 183					el_coeff)		7							
	67	3	coef_probs_4x4 [0] [0] [0] [1] [0..2]	84, 49, 136	84, 49, 136	0	15												
	70	3	coef_probs_4x4 [0] [0] [0] [2] [0..2]	8, 42, 71	8, 42, 71	0	18												
	73	3	coef_probs_4x4 [0] [0] [1] [0] [0..2]	31, 107, 169	31, 107, 169	0	21												
	76	3	coef_probs_4x4 [0] [0] [1] [1] [0..2]	35, 99, 159	35, 99, 159	0	24												
	79	3	coef_probs_4x4 [0] [0] [1] [2] [0..2]	17, 82, 140	17, 82, 140	0	27												
	82	3	coef_probs_4x4 [0] [0] [1] [3] [0..2]	8, 66, 114	8, 66, 11	0	30												

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	85	3	coef_probs_4x4 [0] [0] [1] [4] [0..2]	2, 44, 76	2, 44, 76	0	33											
	88	3	coef_probs_4x4 [0] [0] [1] [5] [0..2]	1, 19, 32	1, 19, 32	0	36											
	91	3	coef_probs_4x4 [0] [0] [2] [0] [0..2]	40, 132, 201	40, 132, 201	0	39											
	94	3	coef_probs_4x4 [0] [0] [2] [1] [0..2]	29, 114, 187	29, 114, 187	0	42											
	97	3	coef_probs_4x4 [0] [0] [2] [2] [0..2]	13, 91, 157	13, 91, 157	0	45											
	100	3	coef_probs_4x4 [0] [0] [2] [3] [0..2]	7, 75, 127	7, 75, 127	0	48											
	103	3	coef_probs_4x4	3, 58,	3,	0	51											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [0] [2] [4] [0..2]	95	58, 95													
	106	3	coef_probs_4x4 [0] [0] [2] [5] [0..2]	1, 28, 47	1, 28, 47	0	54											
	109	3	coef_probs_4x4 [0] [0] [3] [0] [0..2]	69, 142, 221	69, 142, 221	0	57											
	112	3	coef_probs_4x4 [0] [0] [3] [1] [0..2]	42, 122, 201	42, 122, 201	0	60											
	115	3	coef_probs_4x4 [0] [0] [3] [2] [0..2]	15, 91, 159	15, 91, 159	0	63											
	118	3	coef_probs_4x4 [0] [0] [3] [3] [0..2]	6, 67, 121	6, 67, 121	0	66											
	121	3	coef_probs_4x4 [0] [0] [3] [4] [0..2]	1, 42, 77	1, 42, 77	0	69											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	124	3	coef_probs_4x4 [0] [0] [3] [5] [0..2]	1, 17, 31	1, 17, 31	0	72											
	127	3	coef_probs_4x4 [0] [0] [4] [0] [0..2]	102, 148, 228	102, 148, 228	0	75											
	130	3	coef_probs_4x4 [0] [0] [4] [1] [0..2]	67, 117, 204	67, 117, 204	0	78											
	133	3	coef_probs_4x4 [0] [0] [4] [2] [0..2]	17, 82, 154	17, 82, 154	0	81											
	136	3	coef_probs_4x4 [0] [0] [4] [3] [0..2]	6, 59, 114	6, 59, 114	0	84											
	139	3	coef_probs_4x4 [0] [0] [4] [4] [0..2]	2, 39, 75	2, 39, 75	0	87											
	142	3	coef_probs_4x4	1, 15,	1,	0	90											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [0] [4] [5] [0..2]	29	15, 29													
	145	3	coef_probs_4x4 [0] [0] [5] [0] [0..2]	156, 57, 233	156, 57, 233	0	93											
	148	3	coef_probs_4x4 [0] [0] [5] [1] [0..2]	119, 57, 212	119, 57, 212	0	96											
	151	3	coef_probs_4x4 [0] [0] [5] [2] [0..2]	58, 48, 163	58, 48, 163	0	99											
	154	3	coef_probs_4x4 [0] [0] [5] [3] [0..2]	29, 40, 124	29, 40, 124	0	102											
	157	3	coef_probs_4x4 [0] [0] [5] [4] [0..2]	12, 30, 81	12, 30, 81	0	105											
	160	3	coef_probs_4x4 [0] [0] [5] [5] [0..2]	3, 12, 31	3, 12, 31	0	108											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	163	3	coef_probs_4x4 [0] [1] [0] [0] [0..2]	191, 107, 226	191, 107, 226	0	111											
	166	3	coef_probs_4x4 [0] [1] [0] [1] [0..2]	124, 117, 204	124, 117, 204	0	114											
	169	3	coef_probs_4x4 [0] [1] [0] [2] [0..2]	25, 99, 155	25, 99, 155	0	117											
	172	3	coef_probs_4x4 [0] [1] [1] [0] [0..2]	29, 148, 210	29, 148, 210	0	120											
	175	3	coef_probs_4x4 [0] [1] [1] [1] [0..2]	37, 126, 194	37, 126, 194	0	123											
	178	3	coef_probs_4x4	8, 93,	8,	0	12											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [1] [1] [2] [0..2]	157	93, 157		6											
	181	3	coef_probs_4x4 [0] [1] [1] [3] [0..2]	2, 68, 118	2, 68, 118	0	129											
	184	3	coef_probs_4x4 [0] [1] [1] [4] [0..2]	1, 39, 69	1, 39, 69	0	132											
	187	3	coef_probs_4x4 [0] [1] [1] [5] [0..2]	1, 17, 33	1, 17, 33	0	135											
	190	3	coef_probs_4x4 [0] [1] [2] [0] [0..2]	41, 151, 213	41, 151, 213	0	138											
	193	3	coef_probs_4x4 [0] [1] [2] [1] [0..2]	27, 123, 193	27, 123, 193	0	141											
	196	3	coef_probs_4x4 [0] [1] [2] [2] [0..2]	3, 82, 144	3, 82, 14	0	144											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults					Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4														
	199	3	coef_probs_4x4 [0] [1] [2] [3] [0..2]	1, 58, 105	1, 58, 105	0	147												
	202	3	coef_probs_4x4 [0] [1] [2] [4] [0..2]	1, 32, 60	1, 32, 60	0	150												
	205	3	coef_probs_4x4 [0] [1] [2] [5] [0..2]	1, 13, 26	1, 13, 26	0	153												
	208	3	coef_probs_4x4 [0] [1] [3] [0] [0..2]	59, 159, 220	59, 159, 220	0	156												
	211	3	coef_probs_4x4 [0] [1] [3] [1] [0..2]	23, 126, 198	23, 126, 198	0	159												
	214	3	coef_probs_4x4 [0] [1] [3] [2] [0..2]	4, 88, 151	4, 88, 151	0	162												
	217	3	coef_probs_4x4	1, 66,	1,	0	16												

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [1] [3] [3] [0..2]	114	66, 114		5											
	220	3	coef_probs_4x4 [0] [1] [3] [4] [0..2]	1, 38, 71	1, 38, 71	0	168											
	223	3	coef_probs_4x4 [0] [1] [3] [5] [0..2]	1, 18, 34	1, 18, 34	0	171											
	226	3	coef_probs_4x4 [0] [1] [4] [0] [0..2]	114, 136, 232	114, 136, 232	0	174											
	229	3	coef_probs_4x4 [0] [1] [4] [1] [0..2]	51, 114, 207	51, 114, 207	0	177											
	232	3	coef_probs_4x4 [0] [1] [4] [2] [0..2]	11, 83, 155	11, 83, 155	0	180											
	235	3	coef_probs_4x4 [0] [1] [4] [3]	3, 56, 105	3, 56,	0	183											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		105													
	238	3	coef_probs_4x4 [0] [1] [4] [4] [0..2]	1, 33, 65	1, 33, 65	0	186											
	241	3	coef_probs_4x4 [0] [1] [4] [5] [0..2]	1, 17, 34	1, 17, 34	0	189											
	244	3	coef_probs_4x4 [0] [1] [5] [0] [0..2]	149, 65, 234	149, 65, 234	0	192											
	247	3	coef_probs_4x4 [0] [1] [5] [1] [0..2]	121, 57, 215	121, 57, 215	0	195											
	250	3	coef_probs_4x4 [0] [1] [5] [2] [0..2]	61, 49, 166	61, 49, 166	0	198											
	253	3	coef_probs_4x4 [0] [1] [5] [3] [0..2]	28, 36, 114	28, 36, 114	0	201											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	256	3	coef_probs_4x4 [0] [1] [5] [4] [0..2]	12, 25, 76	12, 25, 76	0	204											
	259	3	coef_probs_4x4 [0] [1] [5] [5] [0..2]	3, 16, 42	3, 16, 42	0	207											
	262	3	coef_probs_4x4 [1] [0] [0] [0] [0..2]	214, 49, 220	214, 49, 220	0	210					0-287						
	265	3	coef_probs_4x4 [1] [0] [0] [1] [0..2]	132, 63, 188	132, 63, 188	0	213											
	268	3	coef_probs_4x4 [1] [0] [0] [2] [0..2]	42, 65, 137	42, 65, 137	0	216											
	271	3	coef_probs_4x4 [1] [0] [1] [0] [0..2]	85, 137, 221	85, 137, 221	0	219											
	274	3	coef_probs_4x4	104,	10	0	22											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [0] [1] [1] [0..2]	131, 216	4, 131, 216		2											
	277	3	coef_probs_4x4 [1] [0] [1] [2] [0..2]	49, 111, 192	49, 111, 192	0	225											
	280	3	coef_probs_4x4 [1] [0] [1] [3] [0..2]	21, 87, 155	21, 87, 155	0	228											
	283	3	coef_probs_4x4 [1] [0] [1] [4] [0..2]	2, 49, 87	2, 49, 87	0	231											
	286	3	coef_probs_4x4 [1] [0] [1] [5] [0..2]	1, 16, 28	1, 16, 28	0	234											
	289	3	coef_probs_4x4 [1] [0] [2] [0] [0..2]	89, 163, 230	89, 163, 230	0	237											
	292	3	coef_probs_4x4	90,	90,	0	24											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [0] [2] [1] [0..2]	137, 220	137, 220		0											
	295	3	coef_probs_4x4 [1] [0] [2] [2] [0..2]	29, 100, 183	29, 100, 183	0	243											
	298	3	coef_probs_4x4 [1] [0] [2] [3] [0..2]	10, 70, 135	10, 70, 135	0	246											
	301	3	coef_probs_4x4 [1] [0] [2] [4] [0..2]	2, 42, 81	2, 42, 81	0	249											
	304	3	coef_probs_4x4 [1] [0] [2] [5] [0..2]	1, 17, 33	1, 17, 33	0	252											
	307	3	coef_probs_4x4 [1] [0] [3] [0] [0..2]	108, 167, 237	108, 167, 237	0	255											
	310	3	coef_probs_4x4	55,	55,	0	25											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [0] [3] [1] [0..2]	133, 222	133, 222		8											
	313	3	coef_probs_4x4 [1] [0] [3] [2] [0..2]	15, 97, 179	15, 97, 179	0	261											
	316	3	coef_probs_4x4 [1] [0] [3] [3] [0..2]	4, 72, 135	4, 72, 135	0	264											
	319	3	coef_probs_4x4 [1] [0] [3] [4] [0..2]	1, 45, 85	1, 45, 85	0	267											
	322	3	coef_probs_4x4 [1] [0] [3] [5] [0..2]	1, 19, 38	1, 19, 38	0	270											
	325	3	coef_probs_4x4 [1] [0] [4] [0] [0..2]	124, 146, 240	124, 146, 240	0	273											
	328	3	coef_probs_4x4 [1] [0] [4] [1]	66, 124,	66, 12	0	276											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]	224	4, 224													
	331	3	coef_probs_4x4 [1] [0] [4] [2] [0..2]	17, 88, 175	17, 88, 175	0	279											
	334	3	coef_probs_4x4 [1] [0] [4] [3] [0..2]	4, 58, 122	4, 58, 122	0	282											
	337	3	coef_probs_4x4 [1] [0] [4] [4] [0..2]	1, 36, 75	1, 36, 75	0	285											
	340	3	coef_probs_4x4 [1] [0] [4] [5] [0..2]	1, 18, 37	1, 18, 37	0	288											
	343	3	coef_probs_4x4 [1] [0] [5] [0] [0..2]	141, 79, 241	141, 79, 241	0	291											
	346	3	coef_probs_4x4 [1] [0] [5] [1] [0..2]	126, 70, 227	126, 70, 22	0	294											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					7													
	349	3	coef_probs_4x4 [1] [0] [5] [2] [0..2]	66, 58, 182	66, 58, 182	0	297											
	352	3	coef_probs_4x4 [1] [0] [5] [3] [0..2]	30, 44, 136	30, 44, 136	0	300											
	355	3	coef_probs_4x4 [1] [0] [5] [4] [0..2]	12, 34, 96	12, 34, 96	0	303											
	358	3	coef_probs_4x4 [1] [0] [5] [5] [0..2]	2, 20, 47	2, 20, 47	0	306											
	361	3	coef_probs_4x4 [1] [1] [0] [0] [0..2]	229, 99, 249	229, 99, 249	0	309											
	364	3	coef_probs_4x4 [1] [1] [0] [1] [0..2]	143, 111, 235	143, 111, 235	0	312											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	367	3	coef_probs_4x4 [1] [1] [0] [2] [0..2]	46, 109, 192	46, 109, 192	0	315											
	370	3	coef_probs_4x4 [1] [1] [1] [0] [0..2]	82, 158, 236	82, 158, 236	0	318											
	373	3	coef_probs_4x4 [1] [1] [1] [1] [0..2]	94, 146, 224	94, 146, 224	0	321											
	376	3	coef_probs_4x4 [1] [1] [1] [2] [0..2]	25, 117, 191	25, 117, 191	0	324											
	379	3	coef_probs_4x4 [1] [1] [1] [3] [0..2]	9, 87, 149	9, 87, 149	0	327											
	382	3	coef_probs_4x4 [1] [1] [1] [4] [0..2]	3, 56, 99	3, 56, 99	0	330											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	385	3	coef_probs_4x4 [1] [1] [1] [5] [0..2]	1, 33, 57	1, 33, 57	0	333											
	388	3	coef_probs_4x4 [1] [1] [2] [0] [0..2]	83, 167, 237	83, 167, 237	0	336											
	391	3	coef_probs_4x4 [1] [1] [2] [1] [0..2]	68, 145, 222	68, 145, 222	0	339											
	394	3	coef_probs_4x4 [1] [1] [2] [2] [0..2]	10, 103, 177	10, 103, 177	0	342											
	397	3	coef_probs_4x4 [1] [1] [2] [3] [0..2]	2, 72, 131	2, 72, 131	0	345											
	400	3	coef_probs_4x4 [1] [1] [2] [4] [0..2]	1, 41, 79	1, 41, 79	0	348											
	403	3	coef_probs_4x4	1, 20,	1,	0	35											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults			Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
										4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [2] [5] [0..2]	39	20, 39		1											
	406	3	coef_probs_4x4 [1] [1] [3] [0] [0..2]	99, 167, 239	99, 167, 239	0	354											
	409	3	coef_probs_4x4 [1] [1] [3] [1] [0..2]	47, 141, 224	47, 141, 224	0	357											
	412	3	coef_probs_4x4 [1] [1] [3] [2] [0..2]	10, 104, 178	10, 104, 178	0	360											
	415	3	coef_probs_4x4 [1] [1] [3] [3] [0..2]	2, 73, 133	2, 73, 133	0	363											
	418	3	coef_probs_4x4 [1] [1] [3] [4] [0..2]	1, 44, 85	1, 44, 85	0	366											
	421	3	coef_probs_4x4 [1] [1] [3] [5]	1, 22, 47	1, 22,	0	369											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		47													
	424	3	coef_probs_4x4 [1] [1] [4] [0] [0..2]	127, 145, 243	127, 145, 243	0	372											
	427	3	coef_probs_4x4 [1] [1] [4] [1] [0..2]	71, 129, 228	71, 129, 228	0	375											
	430	3	coef_probs_4x4 [1] [1] [4] [2] [0..2]	17, 93, 177	17, 93, 177	0	378											
	433	3	coef_probs_4x4 [1] [1] [4] [3] [0..2]	3, 61, 124	3, 61, 124	0	381											
	436	3	coef_probs_4x4 [1] [1] [4] [4] [0..2]	1, 41, 84	1, 41, 84	0	384											
	439	3	coef_probs_4x4 [1] [1] [4] [5] [0..2]	1, 21, 52	1, 21, 52	0	387											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	442	3	coef_probs_4x4 [1] [1] [5] [0] [0..2]	157, 78, 244	157, 78, 244	0	390											
	445	3	coef_probs_4x4 [1] [1] [5] [1] [0..2]	140, 72, 231	140, 72, 231	0	393											
	448	3	coef_probs_4x4 [1] [1] [5] [2] [0..2]	69, 58, 184	69, 58, 184	0	396											
	451	3	coef_probs_4x4 [1] [1] [5] [3] [0..2]	31, 44, 137	31, 44, 137	0	399											
	454	3	coef_probs_4x4 [1] [1] [5] [4] [0..2]	14, 38, 105	14, 38, 105	0	402											
	457	3	coef_probs_4x4 [1] [1] [5] [5] [0..2]	8, 23, 61	8, 23, 61	0	405											
	460	3	coef_probs_8x8	125,	12	0	40		57.				0-					

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [0] [0] [0] [0..2]	34, 187	5, 34, 187		8		5				287					
	463	3	coef_probs_8x8 [0] [0] [0] [1] [0..2]	52, 41, 133	52, 41, 133	0	411											
	466	3	coef_probs_8x8 [0] [0] [0] [2] [0..2]	6, 31, 56	6, 31, 56	0	414											
	469	3	coef_probs_8x8 [0] [0] [1] [0] [0..2]	37, 109, 153	37, 109, 153	0	417											
	472	3	coef_probs_8x8 [0] [0] [1] [1] [0..2]	51, 102, 147	51, 102, 147	0	420											
	475	3	coef_probs_8x8 [0] [0] [1] [2] [0..2]	23, 87, 128	23, 87, 128	0	423											
	478	3	coef_probs_8x8	8, 67,	8,	0	42											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [0] [1] [3] [0..2]	101	67, 101		6											
	481	3	coef_probs_8x8 [0] [0] [1] [4] [0..2]	1, 41, 63	1, 41, 63	0	429											
	484	3	coef_probs_8x8 [0] [0] [1] [5] [0..2]	1, 19, 29	1, 19, 29	0	432											
	487	3	coef_probs_8x8 [0] [0] [2] [0] [0..2]	31, 154, 185	31, 154, 185	0	435											
	490	3	coef_probs_8x8 [0] [0] [2] [1] [0..2]	17, 127, 175	17, 127, 175	0	438											
	493	3	coef_probs_8x8 [0] [0] [2] [2] [0..2]	6, 96, 145	6, 96, 145	0	441											
	496	3	coef_probs_8x8 [0] [0] [2] [3] [0..2]	2, 73, 114	2, 73, 11	0	444											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	499	3	coef_probs_8x8 [0] [0] [2] [4] [0..2]	1, 51, 82	1, 51, 82	0	447											
	502	3	coef_probs_8x8 [0] [0] [2] [5] [0..2]	1, 28, 45	1, 28, 45	0	450											
	505	3	coef_probs_8x8 [0] [0] [3] [0] [0..2]	23, 163, 200	23, 163, 200	0	453											
	508	3	coef_probs_8x8 [0] [0] [3] [1] [0..2]	10, 131, 185	10, 131, 185	0	456											
	511	3	coef_probs_8x8 [0] [0] [3] [2] [0..2]	2, 93, 148	2, 93, 148	0	459											
	514	3	coef_probs_8x8 [0] [0] [3] [3] [0..2]	1, 67, 111	1, 67, 111	0	462											
	517	3	coef_probs_8x8	1, 41,	1,	0	46											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [0] [3] [4] [0..2]	69	41, 69		5											
	520	3	coef_probs_8x8 [0] [0] [3] [5] [0..2]	1, 14, 24	1, 14, 24	0	468											
	523	3	coef_probs_8x8 [0] [0] [4] [0] [0..2]	29, 176, 217	29, 176, 217	0	471											
	526	3	coef_probs_8x8 [0] [0] [4] [1] [0..2]	12, 145, 201	12, 145, 201	0	474											
	529	3	coef_probs_8x8 [0] [0] [4] [2] [0..2]	3, 101, 156	3, 101, 156	0	477											
	532	3	coef_probs_8x8 [0] [0] [4] [3] [0..2]	1, 69, 111	1, 69, 111	0	480											
	535	3	coef_probs_8x8 [0] [0] [4] [4]	1, 39, 63	1, 39,	0	483											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		63													
	538	3	coef_probs_8x8 [0] [0] [4] [5] [0..2]	1, 14, 23	1, 14, 23	0	486											
	541	3	coef_probs_8x8 [0] [0] [5] [0] [0..2]	57, 192, 233	57, 192, 233	0	489											
	544	3	coef_probs_8x8 [0] [0] [5] [1] [0..2]	25, 154, 215	25, 154, 215	0	492											
	547	3	coef_probs_8x8 [0] [0] [5] [2] [0..2]	6, 109, 167	6, 109, 167	0	495											
	550	3	coef_probs_8x8 [0] [0] [5] [3] [0..2]	3, 78, 118	3, 78, 118	0	498											
	553	3	coef_probs_8x8 [0] [0] [5] [4] [0..2]	1, 48, 69	1, 48, 69	0	501											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	556	3	coef_probs_8x8 [0] [0] [5] [5] [0..2]	1, 21, 29	1, 21, 29	0	504											
	559	3	coef_probs_8x8 [0] [1] [0] [0] [0..2]	202, 105, 245	202, 105, 245	0	507											
	562	3	coef_probs_8x8 [0] [1] [0] [1] [0..2]	108, 106, 216	108, 106, 216	0	510											
	565	3	coef_probs_8x8 [0] [1] [0] [2] [0..2]	18, 90, 144	18, 90, 144	0	513											
	568	3	coef_probs_8x8 [0] [1] [1] [0] [0..2]	33, 172, 219	33, 172, 219	0	516											
	571	3	coef_probs_8x8 [0] [1] [1] [1] [0..2]	64, 149, 206	64, 149,	0	519											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					206													
	574	3	coef_probs_8x8 [0] [1] [1] [2] [0..2]	14, 117, 177	14, 117, 177	0	522											
	577	3	coef_probs_8x8 [0] [1] [1] [3] [0..2]	5, 90, 141	5, 90, 141	0	525											
	580	3	coef_probs_8x8 [0] [1] [1] [4] [0..2]	2, 61, 95	2, 61, 95	0	528											
	583	3	coef_probs_8x8 [0] [1] [1] [5] [0..2]	1, 37, 57	1, 37, 57	0	531											
	586	3	coef_probs_8x8 [0] [1] [2] [0] [0..2]	33, 179, 220	33, 179, 220	0	534											
	589	3	coef_probs_8x8 [0] [1] [2] [1] [0..2]	11, 140, 198	11, 140, 19	0	537											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					8													
	592	3	coef_probs_8x8 [0] [1] [2] [2] [0..2]	1, 89, 148	1, 89, 148	0	540											
	595	3	coef_probs_8x8 [0] [1] [2] [3] [0..2]	1, 60, 104	1, 60, 104	0	543											
	598	3	coef_probs_8x8 [0] [1] [2] [4] [0..2]	1, 33, 57	1, 33, 57	0	546											
	601	3	coef_probs_8x8 [0] [1] [2] [5] [0..2]	1, 12, 21	1, 12, 21	0	549											
	604	3	coef_probs_8x8 [0] [1] [3] [0] [0..2]	30, 181, 221	30, 181, 221	0	552											
	607	3	coef_probs_8x8 [0] [1] [3] [1] [0..2]	8, 141, 198	8, 141, 198	0	555											
	610	3	coef_probs_8x8	1, 87,	1,	0	55											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [1] [3] [2] [0..2]	145	87, 145		8											
	613	3	coef_probs_8x8 [0] [1] [3] [3] [0..2]	1, 58, 100	1, 58, 100	0	561											
	616	3	coef_probs_8x8 [0] [1] [3] [4] [0..2]	1, 31, 55	1, 31, 55	0	564											
	619	3	coef_probs_8x8 [0] [1] [3] [5] [0..2]	1, 12, 20	1, 12, 20	0	567											
	622	3	coef_probs_8x8 [0] [1] [4] [0] [0..2]	32, 186, 224	32, 186, 224	0	570											
	625	3	coef_probs_8x8 [0] [1] [4] [1] [0..2]	7, 142, 198	7, 142, 198	0	573											
	628	3	coef_probs_8x8 [0] [1] [4] [2] [0..2]	1, 86, 143	1, 86, 14	0	576											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					3													
	631	3	coef_probs_8x8 [0] [1] [4] [3] [0..2]	1, 58, 100	1, 58, 100	0	579											
	634	3	coef_probs_8x8 [0] [1] [4] [4] [0..2]	1, 31, 55	1, 31, 55	0	582											
	637	3	coef_probs_8x8 [0] [1] [4] [5] [0..2]	1, 12, 22	1, 12, 22	0	585											
	640	3	coef_probs_8x8 [0] [1] [5] [0] [0..2]	57, 192, 227	57, 192, 227	0	588											
	643	3	coef_probs_8x8 [0] [1] [5] [1] [0..2]	20, 143, 204	20, 143, 204	0	591											
	646	3	coef_probs_8x8 [0] [1] [5] [2] [0..2]	3, 96, 154	3, 96, 154	0	594											
	649	3	coef_probs_8x8	1, 68,	1,	0	59											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [1] [5] [3] [0..2]	112	68, 112		7											
	652	3	coef_probs_8x8 [0] [1] [5] [4] [0..2]	1, 42, 69	1, 42, 69	0	600											
	655	3	coef_probs_8x8 [0] [1] [5] [5] [0..2]	1, 19, 32	1, 19, 32	0	603											
	658	3	coef_probs_8x8 [1] [0] [0] [0] [0..2]	212, 35, 215	212, 35, 215	0	606								0-287			
	661	3	coef_probs_8x8 [1] [0] [0] [1] [0..2]	113, 47, 169	113, 47, 169	0	609											
	664	3	coef_probs_8x8 [1] [0] [0] [2] [0..2]	29, 48, 105	29, 48, 105	0	612											
	667	3	coef_probs_8x8 [1] [0] [1] [0] [0..2]	74, 129, 203	74, 129,	0	615											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					203													
	670	3	coef_probs_8x8 [1] [0] [1] [1] [0..2]	106, 120, 203	106, 120, 203	0	618											
	673	3	coef_probs_8x8 [1] [0] [1] [2] [0..2]	49, 107, 178	49, 107, 178	0	621											
	676	3	coef_probs_8x8 [1] [0] [1] [3] [0..2]	19, 84, 144	19, 84, 144	0	624											
	679	3	coef_probs_8x8 [1] [0] [1] [4] [0..2]	4, 50, 84	4, 50, 84	0	627											
	682	3	coef_probs_8x8 [1] [0] [1] [5] [0..2]	1, 15, 25	1, 15, 25	0	630											
	685	3	coef_probs_8x8 [1] [0] [2] [0] [0..2]	71, 172, 217	71, 172,	0	633											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					217													
	688	3	coef_probs_8x8 [1] [0] [2] [1] [0..2]	44, 141, 209	44, 141, 209	0	636											
	691	3	coef_probs_8x8 [1] [0] [2] [2] [0..2]	15, 102, 173	15, 102, 173	0	639											
	694	3	coef_probs_8x8 [1] [0] [2] [3] [0..2]	6, 76, 133	6, 76, 133	0	642											
	697	3	coef_probs_8x8 [1] [0] [2] [4] [0..2]	2, 51, 89	2, 51, 89	0	645											
	700	3	coef_probs_8x8 [1] [0] [2] [5] [0..2]	1, 24, 42	1, 24, 42	0	648											
	703	3	coef_probs_8x8 [1] [0] [3] [0] [0..2]	64, 185, 231	64, 185, 23	0	651											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					1													
	706	3	coef_probs_8x8 [1] [0] [3] [1] [0..2]	31, 148, 216	31, 148, 216	0	654											
	709	3	coef_probs_8x8 [1] [0] [3] [2] [0..2]	8, 103, 175	8, 103, 175	0	657											
	712	3	coef_probs_8x8 [1] [0] [3] [3] [0..2]	3, 74, 131	3, 74, 131	0	660											
	715	3	coef_probs_8x8 [1] [0] [3] [4] [0..2]	1, 46, 81	1, 46, 81	0	663											
	718	3	coef_probs_8x8 [1] [0] [3] [5] [0..2]	1, 18, 30	1, 18, 30	0	666											
	721	3	coef_probs_8x8 [1] [0] [4] [0] [0..2]	65, 196, 235	65, 196, 235	0	669											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	724	3	coef_probs_8x8 [1] [0] [4] [1] [0..2]	25, 157, 221	25, 157, 221	0	672											
	727	3	coef_probs_8x8 [1] [0] [4] [2] [0..2]	5, 105, 174	5, 105, 174	0	675											
	730	3	coef_probs_8x8 [1] [0] [4] [3] [0..2]	1, 67, 120	1, 67, 120	0	678											
	733	3	coef_probs_8x8 [1] [0] [4] [4] [0..2]	1, 38, 69	1, 38, 69	0	681											
	736	3	coef_probs_8x8 [1] [0] [4] [5] [0..2]	1, 15, 30	1, 15, 30	0	684											
	739	3	coef_probs_8x8 [1] [0] [5] [0] [0..2]	65, 204, 238	65, 204, 238	0	687											
	742	3	coef_probs_8x8	30,	30,	0	69											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [0] [5] [1] [0..2]	156, 224	156, 224		0											
	745	3	coef_probs_8x8 [1] [0] [5] [2] [0..2]	7, 107, 177	7, 107, 177	0	693											
	748	3	coef_probs_8x8 [1] [0] [5] [3] [0..2]	2, 70, 124	2, 70, 124	0	696											
	751	3	coef_probs_8x8 [1] [0] [5] [4] [0..2]	1, 42, 73	1, 42, 73	0	699											
	754	3	coef_probs_8x8 [1] [0] [5] [5] [0..2]	1, 18, 34	1, 18, 34	0	702											
	757	3	coef_probs_8x8 [1] [1] [0] [0] [0..2]	225, 86, 251	225, 86, 251	0	705											
	760	3	coef_probs_8x8 [1] [1] [0] [1]	144, 104,	144,	0	708											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]	235	104, 235													
	763	3	coef_probs_8x8 [1] [1] [0] [2] [0..2]	42, 99, 181	42, 99, 181	0	711											
	766	3	coef_probs_8x8 [1] [1] [1] [0] [0..2]	85, 175, 239	85, 175, 239	0	714											
	769	3	coef_probs_8x8 [1] [1] [1] [1] [0..2]	112, 165, 229	112, 165, 229	0	717											
	772	3	coef_probs_8x8 [1] [1] [1] [2] [0..2]	29, 136, 200	29, 136, 200	0	720											
	775	3	coef_probs_8x8 [1] [1] [1] [3] [0..2]	12, 103, 162	12, 103,	0	723											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					162													
	778	3	coef_probs_8x8 [1] [1] [1] [4] [0..2]	6, 77, 123	6, 77, 123	0	726											
	781	3	coef_probs_8x8 [1] [1] [1] [5] [0..2]	2, 53, 84	2, 53, 84	0	729											
	784	3	coef_probs_8x8 [1] [1] [2] [0] [0..2]	75, 183, 239	75, 183, 239	0	732											
	787	3	coef_probs_8x8 [1] [1] [2] [1] [0..2]	30, 155, 221	30, 155, 221	0	735											
	790	3	coef_probs_8x8 [1] [1] [2] [2] [0..2]	3, 106, 171	3, 106, 171	0	738											
	793	3	coef_probs_8x8 [1] [1] [2] [3]	1, 74, 128	1, 74,	0	741											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		128													
	796	3	coef_probs_8x8 [1] [1] [2] [4] [0..2]	1, 44, 76	1, 44, 76	0	744											
	799	3	coef_probs_8x8 [1] [1] [2] [5] [0..2]	1, 17, 28	1, 17, 28	0	747											
	802	3	coef_probs_8x8 [1] [1] [3] [0] [0..2]	73, 185, 240	73, 185, 240	0	750											
	805	3	coef_probs_8x8 [1] [1] [3] [1] [0..2]	27, 159, 222	27, 159, 222	0	753											
	808	3	coef_probs_8x8 [1] [1] [3] [2] [0..2]	2, 107, 172	2, 107, 172	0	756											
	811	3	coef_probs_8x8 [1] [1] [3] [3] [0..2]	1, 75, 127	1, 75, 12	0	759											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					7													
	814	3	coef_probs_8x8 [1] [1] [3] [4] [0..2]	1, 42, 73	1, 42, 73	0	762											
	817	3	coef_probs_8x8 [1] [1] [3] [5] [0..2]	1, 17, 29	1, 17, 29	0	765											
	820	3	coef_probs_8x8 [1] [1] [4] [0] [0..2]	62, 190, 238	62, 190, 238	0	768											
	823	3	coef_probs_8x8 [1] [1] [4] [1] [0..2]	21, 159, 222	21, 159, 222	0	771											
	826	3	coef_probs_8x8 [1] [1] [4] [2] [0..2]	2, 107, 172	2, 107, 172	0	774											
	829	3	coef_probs_8x8 [1] [1] [4] [3] [0..2]	1, 72, 122	1, 72, 122	0	777											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	832	3	coef_probs_8x8 [1] [1] [4] [4] [0..2]	1, 40, 71	1, 40, 71	0	780											
	835	3	coef_probs_8x8 [1] [1] [4] [5] [0..2]	1, 18, 32	1, 18, 32	0	783											
	838	3	coef_probs_8x8 [1] [1] [5] [0] [0..2]	61, 199, 240	61, 199, 240	0	786											
	841	3	coef_probs_8x8 [1] [1] [5] [1] [0..2]	27, 161, 226	27, 161, 226	0	789											
	844	3	coef_probs_8x8 [1] [1] [5] [2] [0..2]	4, 113, 180	4, 113, 180	0	792											
	847	3	coef_probs_8x8 [1] [1] [5] [3] [0..2]	1, 76, 129	1, 76, 129	0	795											
	850	3	coef_probs_8x8	1, 46,	1,	0	79											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [5] [4] [0..2]	80	46, 80		8											
	853	3	coef_probs_8x8 [1] [1] [5] [5] [0..2]	1, 23, 41	1, 23, 41	0	801											
	856	3	coef_probs_16x16 [0] [0] [0] [0] [0..2]	7, 27, 153	7, 27, 153	0	804		107						0-287			
	859	3	coef_probs_16x16 [0] [0] [0] [1] [0..2]	5, 30, 95	5, 30, 95	0	807											
	862	3	coef_probs_16x16 [0] [0] [0] [2] [0..2]	1, 16, 30	1, 16, 30	0	810											
	865	3	coef_probs_16x16 [0] [0] [1] [0] [0..2]	50, 75, 127	50, 75, 127	0	813											
	868	3	coef_probs_16x16 [0] [0] [1] [1] [0..2]	57, 75, 124	57, 75, 124	0	816											
	871	3	coef_probs_16x1	27, 67,	27,	0	81											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults			Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
										4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			6 [0] [0] [1] [2] [0..2]	108	67, 108		9											
	874	3	coef_probs_16x16 [0] [0] [1] [3] [0..2]	10, 54, 86	10, 54, 86	0	822											
	877	3	coef_probs_16x16 [0] [0] [1] [4] [0..2]	1, 33, 52	1, 33, 52	0	825											
	880	3	coef_probs_16x16 [0] [0] [1] [5] [0..2]	1, 12, 18	1, 12, 18	0	828											
	883	3	coef_probs_16x16 [0] [0] [2] [0] [0..2]	43, 125, 151	43, 125, 151	0	831											
	886	3	coef_probs_16x16 [0] [0] [2] [1] [0..2]	26, 108, 148	26, 108, 148	0	834											
	889	3	coef_probs_16x1	7, 83,	7,	0	83											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			6 [0] [0] [2] [2] [0..2]	122	83, 122		7											
	892	3	coef_probs_16x16 [0] [0] [2] [3] [0..2]	2, 59, 89	2, 59, 89	0	840											
	895	3	coef_probs_16x16 [0] [0] [2] [4] [0..2]	1, 38, 60	1, 38, 60	0	843											
	898	3	coef_probs_16x16 [0] [0] [2] [5] [0..2]	1, 17, 27	1, 17, 27	0	846											
	901	3	coef_probs_16x16 [0] [0] [3] [0] [0..2]	23, 144, 163	23, 144, 163	0	849											
	904	3	coef_probs_16x16 [0] [0] [3] [1] [0..2]	13, 112, 154	13, 112, 154	0	852											
	907	3	coef_probs_16x1	2, 75,	2,	0	85											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			6 [0] [0] [3] [2] [0..2]	117	75, 117		5											
	910	3	coef_probs_16x16 [0] [0] [3] [3] [0..2]	1, 50, 81	1, 50, 81	0	858											
	913	3	coef_probs_16x16 [0] [0] [3] [4] [0..2]	1, 31, 51	1, 31, 51	0	861											
	916	3	coef_probs_16x16 [0] [0] [3] [5] [0..2]	1, 14, 23	1, 14, 23	0	864											
	919	3	coef_probs_16x16 [0] [0] [4] [0] [0..2]	18, 162, 185	18, 162, 185	0	867											
	922	3	coef_probs_16x16 [0] [0] [4] [1] [0..2]	6, 123, 171	6, 123, 171	0	870											
	925	3	coef_probs_16x1	1, 78,	1,	0	87											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			6 [0] [0] [4] [2] [0..2]	125	78, 125		3											
	928	3	coef_probs_16x16 [0] [0] [4] [3] [0..2]	1, 51, 86	1, 51, 86	0	876											
	931	3	coef_probs_16x16 [0] [0] [4] [4] [0..2]	1, 31, 54	1, 31, 54	0	879											
	934	3	coef_probs_16x16 [0] [0] [4] [5] [0..2]	1, 14, 23	1, 14, 23	0	882											
	937	3	coef_probs_16x16 [0] [0] [5] [0] [0..2]	15, 199, 227	15, 199, 227	0	885											
	940	3	coef_probs_16x16 [0] [0] [5] [1] [0..2]	3, 150, 204	3, 150, 204	0	888											
	943	3	coef_probs_16x1	1, 91,	1,	0	89											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			6 [0] [0] [5] [2] [0..2]	146	91, 146		1											
	946	3	coef_probs_16x16 [0] [0] [5] [3] [0..2]	1, 55, 95	1, 55, 95	0	894											
	949	3	coef_probs_16x16 [0] [0] [5] [4] [0..2]	1, 30, 53	1, 30, 53	0	897											
	952	3	coef_probs_16x16 [0] [0] [5] [5] [0..2]	1, 11, 20	1, 11, 20	0	900											
	955	3	coef_probs_16x16 [0] [1] [0] [0] [0..2]	19, 55, 240	19, 55, 240	0	903											
	958	3	coef_probs_16x16 [0] [1] [0] [1] [0..2]	19, 59, 196	19, 59, 196	0	906											
	961	3	coef_probs_16x16 [0] [1] [0] [2]	3, 52, 105	3, 52, 10	0	909											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		5													
	964	3	coef_probs_16x16 [0] [1] [1] [0] [0..2]	41, 166, 207	41, 166, 207	0	912											
	967	3	coef_probs_16x16 [0] [1] [1] [1] [0..2]	104, 153, 199	104, 153, 199	0	915											
	970	3	coef_probs_16x16 [0] [1] [1] [2] [0..2]	31, 123, 181	31, 123, 181	0	918											
	973	3	coef_probs_16x16 [0] [1] [1] [3] [0..2]	14, 101, 152	14, 101, 152	0	921											
	976	3	coef_probs_16x16 [0] [1] [1] [4] [0..2]	5, 72, 106	5, 72, 106	0	924											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	979	3	coef_probs_16x16 [0] [1] [1] [5] [0..2]	1, 36, 52	1, 36, 52	0	927											
	982	3	coef_probs_16x16 [0] [1] [2] [0] [0..2]	35, 176, 211	35, 176, 211	0	930											
	985	3	coef_probs_16x16 [0] [1] [2] [1] [0..2]	12, 131, 190	12, 131, 190	0	933											
	988	3	coef_probs_16x16 [0] [1] [2] [2] [0..2]	2, 88, 144	2, 88, 144	0	936											
	991	3	coef_probs_16x16 [0] [1] [2] [3] [0..2]	1, 60, 101	1, 60, 101	0	939											
	994	3	coef_probs_16x16 [0] [1] [2] [4] [0..2]	1, 36, 60	1, 36, 60	0	942											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	997	3	coef_probs_16x16 [0] [1] [2] [5] [0..2]	1, 16, 28	1, 16, 28	0	945											
	1000	3	coef_probs_16x16 [0] [1] [3] [0] [0..2]	28, 183, 213	28, 183, 213	0	948											
	1003	3	coef_probs_16x16 [0] [1] [3] [1] [0..2]	8, 134, 191	8, 134, 191	0	951											
	1006	3	coef_probs_16x16 [0] [1] [3] [2] [0..2]	1, 86, 142	1, 86, 142	0	954											
	1009	3	coef_probs_16x16 [0] [1] [3] [3] [0..2]	1, 56, 96	1, 56, 96	0	957											
	1012	3	coef_probs_16x16 [0] [1] [3] [4] [0..2]	1, 30, 53	1, 30, 53	0	960											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults					Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1015	3	coef_probs_16x16 [0] [1] [3] [5] [0..2]	1, 12, 20	1, 12, 20	0	963												
	1018	3	coef_probs_16x16 [0] [1] [4] [0] [0..2]	20, 190, 215	20, 190, 215	0	966												
	1021	3	coef_probs_16x16 [0] [1] [4] [1] [0..2]	4, 135, 192	4, 135, 192	0	969												
	1024	3	coef_probs_16x16 [0] [1] [4] [2] [0..2]	1, 84, 139	1, 84, 139	0	972												
	1027	3	coef_probs_16x16 [0] [1] [4] [3] [0..2]	1, 53, 91	1, 53, 91	0	975												
	1030	3	coef_probs_16x16 [0] [1] [4] [4] [0..2]	1, 28, 49	1, 28, 49	0	978												

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1033	3	coef_probs_16x16 [0] [1] [4] [5] [0..2]	1, 11, 20	1, 11, 20	0	981											
	1036	3	coef_probs_16x16 [0] [1] [5] [0] [0..2]	13, 196, 216	13, 196, 216	0	984											
	1039	3	coef_probs_16x16 [0] [1] [5] [1] [0..2]	2, 137, 192	2, 137, 192	0	987											
	1042	3	coef_probs_16x16 [0] [1] [5] [2] [0..2]	1, 86, 143	1, 86, 143	0	990											
	1045	3	coef_probs_16x16 [0] [1] [5] [3] [0..2]	1, 57, 99	1, 57, 99	0	993											
	1048	3	coef_probs_16x16 [0] [1] [5] [4] [0..2]	1, 32, 56	1, 32, 56	0	996											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1051	3	coef_probs_16x16 [0] [1] [5] [5] [0..2]	1, 13, 24	1, 13, 24	0	999											
	1054	3	coef_probs_16x16 [1] [0] [0] [0] [0..2]	211, 29, 217	211, 29, 217	0	1002										0-287	
	1057	3	coef_probs_16x16 [1] [0] [0] [1] [0..2]	96, 47, 156	96, 47, 156	0	1005											
	1060	3	coef_probs_16x16 [1] [0] [0] [2] [0..2]	22, 43, 87	22, 43, 87	0	1008											
	1063	3	coef_probs_16x16 [1] [0] [1] [0] [0..2]	78, 120, 193	78, 120, 193	0	1011											
	1066	3	coef_probs_16x16 [1] [0] [1] [1] [0..2]	111, 116, 186	111, 116, 18	0	1014											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					6													
	1069	3	coef_probs_16x16 [1] [0] [1] [2] [0..2]	46, 102, 164	46, 102, 164	0	1017											
	1072	3	coef_probs_16x16 [1] [0] [1] [3] [0..2]	15, 80, 128	15, 80, 128	0	1020											
	1075	3	coef_probs_16x16 [1] [0] [1] [4] [0..2]	2, 49, 76	2, 49, 76	0	1023											
	1078	3	coef_probs_16x16 [1] [0] [1] [5] [0..2]	1, 18, 28	1, 18, 28	0	1026											
	1081	3	coef_probs_16x16 [1] [0] [2] [0] [0..2]	71, 161, 203	71, 161, 203	0	1029											
	1084	3	coef_probs_16x16 [1] [0] [2] [1]	42, 132, 192	42, 132,	0	1032											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		192													
	1087	3	coef_probs_16x16 [1] [0] [2] [2] [0..2]	10, 98, 150	10, 98, 150	0	1035											
	1090	3	coef_probs_16x16 [1] [0] [2] [3] [0..2]	3, 69, 109	3, 69, 109	0	1038											
	1093	3	coef_probs_16x16 [1] [0] [2] [4] [0..2]	1, 44, 70	1, 44, 70	0	1041											
	1096	3	coef_probs_16x16 [1] [0] [2] [5] [0..2]	1, 18, 29	1, 18, 29	0	1044											
	1099	3	coef_probs_16x16 [1] [0] [3] [0] [0..2]	57, 186, 211	57, 186, 211	0	1047											
	1102	3	coef_probs_16x16 [1] [0] [3] [1]	30, 140, 196	30, 140,	0	1050											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		196													
	1105	3	coef_probs_16x16 [1] [0] [3] [2] [0..2]	4, 93, 146	4, 93, 146	0	1053											
	1108	3	coef_probs_16x16 [1] [0] [3] [3] [0..2]	1, 62, 102	1, 62, 102	0	1056											
	1111	3	coef_probs_16x16 [1] [0] [3] [4] [0..2]	1, 38, 65	1, 38, 65	0	1059											
	1114	3	coef_probs_16x16 [1] [0] [3] [5] [0..2]	1, 16, 27	1, 16, 27	0	1062											
	1117	3	coef_probs_16x16 [1] [0] [4] [0] [0..2]	47, 199, 217	47, 199, 217	0	1065											
	1120	3	coef_probs_16x16 [1] [0] [4] [1]	14, 145, 196	14, 145,	0	1068											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		196													
	1123	3	coef_probs_16x16 [1] [0] [4] [2] [0..2]	1, 88, 142	1, 88, 142	0	1071											
	1126	3	coef_probs_16x16 [1] [0] [4] [3] [0..2]	1, 57, 98	1, 57, 98	0	1074											
	1129	3	coef_probs_16x16 [1] [0] [4] [4] [0..2]	1, 36, 62	1, 36, 62	0	1077											
	1132	3	coef_probs_16x16 [1] [0] [4] [5] [0..2]	1, 15, 26	1, 15, 26	0	1080											
	1135	3	coef_probs_16x16 [1] [0] [5] [0] [0..2]	26, 219, 229	26, 219, 229	0	1083											
	1138	3	coef_probs_16x16 [1] [0] [5] [1]	5, 155, 207	5, 155,	0	1086											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		207													
	1141	3	coef_probs_16x16 [1] [0] [5] [2] [0..2]	1, 94, 151	1, 94, 151	0	1089											
	1144	3	coef_probs_16x16 [1] [0] [5] [3] [0..2]	1, 60, 104	1, 60, 104	0	1092											
	1147	3	coef_probs_16x16 [1] [0] [5] [4] [0..2]	1, 36, 62	1, 36, 62	0	1095											
	1150	3	coef_probs_16x16 [1] [0] [5] [5] [0..2]	1, 16, 28	1, 16, 28	0	1098											
	1153	3	coef_probs_16x16 [1] [1] [0] [0] [0..2]	233, 29, 248	233, 29, 248	0	1101											
	1156	3	coef_probs_16x16 [1] [1] [0] [1]	146, 47, 220	146, 47,	0	1104											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		220													
	1159	3	coef_probs_16x16 [1] [1] [0] [2] [0..2]	43, 52, 140	43, 52, 140	0	1107											
	1162	3	coef_probs_16x16 [1] [1] [1] [0] [0..2]	100, 163, 232	100, 163, 232	0	1110											
	1165	3	coef_probs_16x16 [1] [1] [1] [1] [0..2]	179, 161, 222	179, 161, 222	0	1113											
	1168	3	coef_probs_16x16 [1] [1] [1] [2] [0..2]	63, 142, 204	63, 142, 204	0	1116											
	1171	3	coef_probs_16x16 [1] [1] [1] [3] [0..2]	37, 113, 174	37, 113, 17	0	1119											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	1174	3	coef_probs_16x16 [1] [1] [1] [4] [0..2]	26, 89, 137	26, 89, 137	0	1122											
	1177	3	coef_probs_16x16 [1] [1] [1] [5] [0..2]	18, 68, 97	18, 68, 97	0	1125											
	1180	3	coef_probs_16x16 [1] [1] [2] [0] [0..2]	85, 181, 230	85, 181, 230	0	1128											
	1183	3	coef_probs_16x16 [1] [1] [2] [1] [0..2]	32, 146, 209	32, 146, 209	0	1131											
	1186	3	coef_probs_16x16 [1] [1] [2] [2] [0..2]	7, 100, 164	7, 100, 164	0	1134											
	1189	3	coef_probs_16x16	3, 71, 121	3, 71,	0	1137											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [2] [3] [0..2]		121													
	1192	3	coef_probs_16x16 [1] [1] [2] [4] [0..2]	1, 45, 77	1, 45, 77	0	1140											
	1195	3	coef_probs_16x16 [1] [1] [2] [5] [0..2]	1, 18, 30	1, 18, 30	0	1143											
	1198	3	coef_probs_16x16 [1] [1] [3] [0] [0..2]	65, 187, 230	65, 187, 230	0	1146											
	1201	3	coef_probs_16x16 [1] [1] [3] [1] [0..2]	20, 148, 207	20, 148, 207	0	1149											
	1204	3	coef_probs_16x16 [1] [1] [3] [2] [0..2]	2, 97, 159	2, 97, 159	0	1152											
	1207	3	coef_probs_16x16	1, 68, 116	1, 68,	0	1155											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [3] [3] [0..2]		116													
	1210	3	coef_probs_16x16 [1] [1] [3] [4] [0..2]	1, 40, 70	1, 40, 70	0	1158											
	1213	3	coef_probs_16x16 [1] [1] [3] [5] [0..2]	1, 14, 29	1, 14, 29	0	1161											
	1216	3	coef_probs_16x16 [1] [1] [4] [0] [0..2]	40, 194, 227	40, 194, 227	0	1164											
	1219	3	coef_probs_16x16 [1] [1] [4] [1] [0..2]	8, 147, 204	8, 147, 204	0	1167											
	1222	3	coef_probs_16x16 [1] [1] [4] [2] [0..2]	1, 94, 155	1, 94, 155	0	1170											
	1225	3	coef_probs_16x16	1, 65, 112	1, 65,	0	1173											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [4] [3] [0..2]		112													
	1228	3	coef_probs_16x16 [1] [1] [4] [4] [0..2]	1, 39, 66	1, 39, 66	0	1176											
	1231	3	coef_probs_16x16 [1] [1] [4] [5] [0..2]	1, 14, 26	1, 14, 26	0	1179											
	1234	3	coef_probs_16x16 [1] [1] [5] [0] [0..2]	16, 208, 228	16, 208, 228	0	1182											
	1237	3	coef_probs_16x16 [1] [1] [5] [1] [0..2]	3, 151, 207	3, 151, 207	0	1185											
	1240	3	coef_probs_16x16 [1] [1] [5] [2] [0..2]	1, 98, 160	1, 98, 160	0	1188											
	1243	3	coef_probs_16x16	1, 67, 117	1, 67,	0	1191											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [5] [3] [0..2]		117														
	1246	3	coef_probs_16x16 [1] [1] [5] [4] [0..2]	1, 41, 74	1, 41, 74	0	1194												
	1249	3	coef_probs_16x16 [1] [1] [5] [5] [0..2]	1, 17, 31	1, 17, 31	0	1197												
	1252	3	coef_probs_32x32 [0] [0] [0] [0] [0..2]	17, 38, 140	17, 38, 140	0	1200		156.5									0-287	
	1255	3	coef_probs_32x32 [0] [0] [0] [1] [0..2]	7, 34, 80	7, 34, 80	0	1203												
	1258	3	coef_probs_32x32 [0] [0] [0] [2] [0..2]	1, 17, 29	1, 17, 29	0	1206												
	1261	3	coef_probs_32x32 [0] [0] [1] [0] [0..2]	37, 75, 128	37, 75, 128	0	1209												

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1264	3	coef_probs_32x32 [0] [0] [1] [1] [0..2]	41, 76, 128	41, 76, 128	0	1212											
	1267	3	coef_probs_32x32 [0] [0] [1] [2] [0..2]	26, 66, 116	26, 66, 116	0	1215											
	1270	3	coef_probs_32x32 [0] [0] [1] [3] [0..2]	12, 52, 94	12, 52, 94	0	1218											
	1273	3	coef_probs_32x32 [0] [0] [1] [4] [0..2]	2, 32, 55	2, 32, 55	0	1221											
	1276	3	coef_probs_32x32 [0] [0] [1] [5] [0..2]	1, 10, 16	1, 10, 16	0	1224											
	1279	3	coef_probs_32x32 [0] [0] [2] [0] [0..2]	50, 127, 154	50, 127, 154	0	1227											
	128	3	coef_probs_32x3	37,	37,	0	12											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	2		2 [0] [0] [2] [1] [0..2]	109, 152	109, 152		30											
	1285	3	coef_probs_32x32 [0] [0] [2] [2] [0..2]	16, 82, 121	16, 82, 121	0	1233											
	1288	3	coef_probs_32x32 [0] [0] [2] [3] [0..2]	5, 59, 85	5, 59, 85	0	1236											
	1291	3	coef_probs_32x32 [0] [0] [2] [4] [0..2]	1, 35, 54	1, 35, 54	0	1239											
	1294	3	coef_probs_32x32 [0] [0] [2] [5] [0..2]	1, 13, 20	1, 13, 20	0	1242											
	1297	3	coef_probs_32x32 [0] [0] [3] [0] [0..2]	40, 142, 167	40, 142, 167	0	1245											
	130	3	coef_probs_32x3	17,	17,	0	12											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	0		2 [0] [0] [3] [1] [0..2]	110, 157	110, 157		48											
	1303	3	coef_probs_32x32 [0] [0] [3] [2] [0..2]	2, 71, 112	2, 71, 112	0	1251											
	1306	3	coef_probs_32x32 [0] [0] [3] [3] [0..2]	1, 44, 72	1, 44, 72	0	1254											
	1309	3	coef_probs_32x32 [0] [0] [3] [4] [0..2]	1, 27, 45	1, 27, 45	0	1257											
	1312	3	coef_probs_32x32 [0] [0] [3] [5] [0..2]	1, 11, 17	1, 11, 17	0	1260											
	1315	3	coef_probs_32x32 [0] [0] [4] [0] [0..2]	30, 175, 188	30, 175, 188	0	1263											
	131	3	coef_probs_32x3	9, 124,	9,	0	12											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	8		2 [0] [0] [4] [1] [0..2]	169	124, 169		66											
	1321	3	coef_probs_32x32 [0] [0] [4] [2] [0..2]	1, 74, 116	1, 74, 116	0	1269											
	1324	3	coef_probs_32x32 [0] [0] [4] [3] [0..2]	1, 48, 78	1, 48, 78	0	1272											
	1327	3	coef_probs_32x32 [0] [0] [4] [4] [0..2]	1, 30, 49	1, 30, 49	0	1275											
	1330	3	coef_probs_32x32 [0] [0] [4] [5] [0..2]	1, 11, 18	1, 11, 18	0	1278											
	1333	3	coef_probs_32x32 [0] [0] [5] [0] [0..2]	10, 222, 223	10, 222, 223	0	1281											
	133	3	coef_probs_32x3	2, 150,	2,	0	12											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	6		2 [0] [0] [5] [1] [0..2]	194	150, 194		84											
	1339	3	coef_probs_32x32 [0] [0] [5] [2] [0..2]	1, 83, 128	1, 83, 128	0	1287											
	1342	3	coef_probs_32x32 [0] [0] [5] [3] [0..2]	1, 48, 79	1, 48, 79	0	1290											
	1345	3	coef_probs_32x32 [0] [0] [5] [4] [0..2]	1, 27, 45	1, 27, 45	0	1293											
	1348	3	coef_probs_32x32 [0] [0] [5] [5] [0..2]	1, 11, 17	1, 11, 17	0	1296											
	1351	3	coef_probs_32x32 [0] [1] [0] [0] [0..2]	36, 41, 235	36, 41, 235	0	1299											
	1354	3	coef_probs_32x32	29, 36, 193	29, 36,	0	1302											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0] [1] [0] [1] [0..2]		193													
	1357	3	coef_probs_32x32 [0] [1] [0] [2] [0..2]	10, 27, 111	10, 27, 111	0	1305											
	1360	3	coef_probs_32x32 [0] [1] [1] [0] [0..2]	85, 165, 222	85, 165, 222	0	1308											
	1363	3	coef_probs_32x32 [0] [1] [1] [1] [0..2]	177, 162, 215	177, 162, 215	0	1311											
	1366	3	coef_probs_32x32 [0] [1] [1] [2] [0..2]	110, 135, 195	110, 135, 195	0	1314											
	1369	3	coef_probs_32x32 [0] [1] [1] [3] [0..2]	57, 113, 168	57, 113, 16	0	1317											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					8													
	1372	3	coef_probs_32x32 [0] [1] [1] [4] [0..2]	23, 83, 120	23, 83, 120	0	1320											
	1375	3	coef_probs_32x32 [0] [1] [1] [5] [0..2]	10, 49, 61	10, 49, 61	0	1323											
	1378	3	coef_probs_32x32 [0] [1] [2] [0] [0..2]	85, 190, 223	85, 190, 223	0	1326											
	1381	3	coef_probs_32x32 [0] [1] [2] [1] [0..2]	36, 139, 200	36, 139, 200	0	1329											
	1384	3	coef_probs_32x32 [0] [1] [2] [2] [0..2]	5, 90, 146	5, 90, 146	0	1332											
	1387	3	coef_probs_32x32 [0] [1] [2] [3]	1, 60, 103	1, 60, 10	0	1335											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		3													
	1390	3	coef_probs_32x32 [0] [1] [2] [4] [0..2]	1, 38, 65	1, 38, 65	0	1338											
	1393	3	coef_probs_32x32 [0] [1] [2] [5] [0..2]	1, 18, 30	1, 18, 30	0	1341											
	1396	3	coef_probs_32x32 [0] [1] [3] [0] [0..2]	72, 202, 223	72, 202, 223	0	1344											
	1399	3	coef_probs_32x32 [0] [1] [3] [1] [0..2]	23, 141, 199	23, 141, 199	0	1347											
	1402	3	coef_probs_32x32 [0] [1] [3] [2] [0..2]	2, 86, 140	2, 86, 140	0	1350											
	1405	3	coef_probs_32x32 [0] [1] [3] [3]	1, 56, 97	1, 56, 97	0	1353											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]															
	1408	3	coef_probs_32x32 [0] [1] [3] [4] [0..2]	1, 36, 61	1, 36, 61	0	1356											
	1411	3	coef_probs_32x32 [0] [1] [3] [5] [0..2]	1, 16, 27	1, 16, 27	0	1359											
	1414	3	coef_probs_32x32 [0] [1] [4] [0] [0..2]	55, 218, 225	55, 218, 225	0	1362											
	1417	3	coef_probs_32x32 [0] [1] [4] [1] [0..2]	13, 145, 200	13, 145, 200	0	1365											
	1420	3	coef_probs_32x32 [0] [1] [4] [2] [0..2]	1, 86, 141	1, 86, 141	0	1368											
	1423	3	coef_probs_32x32 [0] [1] [4] [3]	1, 57, 99	1, 57, 99	0	1371											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]															
	1426	3	coef_probs_32x32 [0] [1] [4] [4] [0..2]	1, 35, 61	1, 35, 61	0	1374											
	1429	3	coef_probs_32x32 [0] [1] [4] [5] [0..2]	1, 13, 22	1, 13, 22	0	1377											
	1432	3	coef_probs_32x32 [0] [1] [5] [0] [0..2]	15, 235, 212	15, 235, 212	0	1380											
	1435	3	coef_probs_32x32 [0] [1] [5] [1] [0..2]	1, 132, 184	1, 132, 184	0	1383											
	1438	3	coef_probs_32x32 [0] [1] [5] [2] [0..2]	1, 84, 139	1, 84, 139	0	1386											
	1441	3	coef_probs_32x32 [0] [1] [5] [3]	1, 57, 97	1, 57, 97	0	1389											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]															
	1444	3	coef_probs_32x32 [0] [1] [5] [4] [0..2]	1, 34, 56	1, 34, 56	0	1392											
	1447	3	coef_probs_32x32 [0] [1] [5] [5] [0..2]	1, 14, 23	1, 14, 23	0	1395											
	1450	3	coef_probs_32x32 [1] [0] [0] [0] [0..2]	181, 21, 201	181, 21, 201	0	1398											0-287
	1453	3	coef_probs_32x32 [1] [0] [0] [1] [0..2]	61, 37, 123	61, 37, 123	0	1401											
	1456	3	coef_probs_32x32 [1] [0] [0] [2] [0..2]	10, 38, 71	10, 38, 71	0	1404											
	1459	3	coef_probs_32x32 [1] [0] [1] [0] [0..2]	47, 106, 172	47, 106, 17	0	1407											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					2													
	1462	3	coef_probs_32x32 [1] [0] [1] [1] [0..2]	95, 104, 173	95, 104, 173	0	1410											
	1465	3	coef_probs_32x32 [1] [0] [1] [2] [0..2]	42, 93, 159	42, 93, 159	0	1413											
	1468	3	coef_probs_32x32 [1] [0] [1] [3] [0..2]	18, 77, 131	18, 77, 131	0	1416											
	1471	3	coef_probs_32x32 [1] [0] [1] [4] [0..2]	4, 50, 81	4, 50, 81	0	1419											
	1474	3	coef_probs_32x32 [1] [0] [1] [5] [0..2]	1, 17, 23	1, 17, 23	0	1422											
	1477	3	coef_probs_32x32 [1] [0] [2] [0] [0..2]	62, 147, 199	62, 147, 19	0	1425											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					9													
	1480	3	coef_probs_32x32 [1] [0] [2] [1] [0..2]	44, 130, 189	44, 130, 189	0	1428											
	1483	3	coef_probs_32x32 [1] [0] [2] [2] [0..2]	28, 102, 154	28, 102, 154	0	1431											
	1486	3	coef_probs_32x32 [1] [0] [2] [3] [0..2]	18, 75, 115	18, 75, 115	0	1434											
	1489	3	coef_probs_32x32 [1] [0] [2] [4] [0..2]	2, 44, 65	2, 44, 65	0	1437											
	1492	3	coef_probs_32x32 [1] [0] [2] [5] [0..2]	1, 12, 19	1, 12, 19	0	1440											
	1495	3	coef_probs_32x32 [1] [0] [3] [0]	55, 153, 210	55, 153,	0	1443											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		210													
	1498	3	coef_probs_32x32 [1] [0] [3] [1] [0..2]	24, 130, 194	24, 130, 194	0	1446											
	1501	3	coef_probs_32x32 [1] [0] [3] [2] [0..2]	3, 93, 146	3, 93, 146	0	1449											
	1504	3	coef_probs_32x32 [1] [0] [3] [3] [0..2]	1, 61, 97	1, 61, 97	0	1452											
	1507	3	coef_probs_32x32 [1] [0] [3] [4] [0..2]	1, 31, 50	1, 31, 50	0	1455											
	1510	3	coef_probs_32x32 [1] [0] [3] [5] [0..2]	1, 10, 16	1, 10, 16	0	1458											
	1513	3	coef_probs_32x32 [1] [0] [4] [0]	49, 186, 223	49, 186,	0	1461											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		223													
	1516	3	coef_probs_32x32 [1] [0] [4] [1] [0..2]	17, 148, 204	17, 148, 204	0	1464											
	1519	3	coef_probs_32x32 [1] [0] [4] [2] [0..2]	1, 96, 142	1, 96, 142	0	1467											
	1522	3	coef_probs_32x32 [1] [0] [4] [3] [0..2]	1, 53, 83	1, 53, 83	0	1470											
	1525	3	coef_probs_32x32 [1] [0] [4] [4] [0..2]	1, 26, 44	1, 26, 44	0	1473											
	1528	3	coef_probs_32x32 [1] [0] [4] [5] [0..2]	1, 11, 17	1, 11, 17	0	1476											
	1531	3	coef_probs_32x32 [1] [0] [5] [0]	13, 217, 212	13, 217,	0	1479											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		212													
	1534	3	coef_probs_32x32 [1] [0] [5] [1] [0..2]	2, 136, 180	2, 136, 180	0	1482											
	1537	3	coef_probs_32x32 [1] [0] [5] [2] [0..2]	1, 78, 124	1, 78, 124	0	1485											
	1540	3	coef_probs_32x32 [1] [0] [5] [3] [0..2]	1, 50, 83	1, 50, 83	0	1488											
	1543	3	coef_probs_32x32 [1] [0] [5] [4] [0..2]	1, 29, 49	1, 29, 49	0	1491											
	1546	3	coef_probs_32x32 [1] [0] [5] [5] [0..2]	1, 14, 23	1, 14, 23	0	1494											
	1549	3	coef_probs_32x32 [1] [1] [0] [0]	197, 13, 247	197, 13,	0	1497											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		247													
	1552	3	coef_probs_32x32 [1] [1] [0] [1] [0..2]	82, 17, 222	82, 17, 222	0	1500											
	1555	3	coef_probs_32x32 [1] [1] [0] [2] [0..2]	25, 17, 162	25, 17, 162	0	1503											
	1558	3	coef_probs_32x32 [1] [1] [1] [0] [0..2]	126, 186, 247	126, 186, 247	0	1506											
	1561	3	coef_probs_32x32 [1] [1] [1] [1] [0..2]	234, 191, 243	234, 191, 243	0	1509											
	1564	3	coef_probs_32x32 [1] [1] [1] [2] [0..2]	176, 177, 234	176, 177, 23	0	1512											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	1567	3	coef_probs_32x32 [1] [1] [1] [3] [0..2]	104, 158, 220	104, 158, 220	0	1515											
	1570	3	coef_probs_32x32 [1] [1] [1] [4] [0..2]	66, 128, 186	66, 128, 186	0	1518											
	1573	3	coef_probs_32x32 [1] [1] [1] [5] [0..2]	55, 90, 137	55, 90, 137	0	1521											
	1576	3	coef_probs_32x32 [1] [1] [2] [0] [0..2]	111, 197, 242	111, 197, 242	0	1524											
	1579	3	coef_probs_32x32 [1] [1] [2] [1] [0..2]	46, 158, 219	46, 158, 219	0	1527											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1582	3	coef_probs_32x32 [1] [1] [2] [2] [0..2]	9, 104, 171	9, 104, 171	0	1530											
	1585	3	coef_probs_32x32 [1] [1] [2] [3] [0..2]	2, 65, 125	2, 65, 125	0	1533											
	1588	3	coef_probs_32x32 [1] [1] [2] [4] [0..2]	1, 44, 80	1, 44, 80	0	1536											
	1591	3	coef_probs_32x32 [1] [1] [2] [5] [0..2]	1, 17, 91	1, 17, 91	0	1539											
	1594	3	coef_probs_32x32 [1] [1] [3] [0] [0..2]	104, 208, 245	104, 208, 245	0	1542											
	1597	3	coef_probs_32x32 [1] [1] [3] [1] [0..2]	39, 168, 224	39, 168, 22	0	1545											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	1600	3	coef_probs_32x32 [1] [1] [3] [2] [0..2]	3, 109, 162	3, 109, 162	0	1548											
	1603	3	coef_probs_32x32 [1] [1] [3] [3] [0..2]	1, 79, 124	1, 79, 124	0	1551											
	1606	3	coef_probs_32x32 [1] [1] [3] [4] [0..2]	1, 50, 102	1, 50, 102	0	1554											
	1609	3	coef_probs_32x32 [1] [1] [3] [5] [0..2]	1, 43, 102	1, 43, 102	0	1557											
	1612	3	coef_probs_32x32 [1] [1] [4] [0] [0..2]	84, 220, 246	84, 220, 246	0	1560											
	1615	3	coef_probs_32x32 [1] [1] [4] [1]	31, 177, 231	31, 177, 7,	0	1563											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[0..2]		231													
	1618	3	coef_probs_32x32 [1] [1] [4] [2] [0..2]	2, 115, 180	2, 115, 180	0	1566											
	1621	3	coef_probs_32x32 [1] [1] [4] [3] [0..2]	1, 79, 134	1, 79, 134	0	1569											
	1624	3	coef_probs_32x32 [1] [1] [4] [4] [0..2]	1, 55, 77	1, 55, 77	0	1572											
	1627	3	coef_probs_32x32 [1] [1] [4] [5] [0..2]	1, 60, 79	1, 60, 79	0	1575											
	1630	3	coef_probs_32x32 [1] [1] [5] [0] [0..2]	43, 243, 240	43, 243, 240	0	1578											
	1633	3	coef_probs_32x32	8, 180, 217	8, 18	0	1581											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
			[1] [1] [5] [1] [0..2]		0, 217													
	1636	3	coef_probs_32x32 [1] [1] [5] [2] [0..2]	1, 115, 166	1, 115, 166	0	1584											
	1639	3	coef_probs_32x32 [1] [1] [5] [3] [0..2]	1, 84, 121	1, 84, 121	0	1587											
	1642	3	coef_probs_32x32 [1] [1] [5] [4] [0..2]	1, 51, 67	1, 51, 67	0	1590											
	1645	3	coef_probs_32x32 [1] [1] [5] [5] [0..2]	1, 16, 6	1, 16, 6	0	1593											
	1648	16	DUMMY	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0,	16	1596											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					0, 0, 0, 0, 0, 0, 0, 0, 0														
CL aligned	1664	3	mbskip_probs [0..2]	192, 128, 64	192, 128, 64	0	1596	12	208	MODE COUNTERS (Others)	18-23								
	1667	3	inter_mode_probs [0] [0..2]	0, 0, 0	2, 173, 34	0	1599	15			24-51								
	1670	3	inter_mode_probs [1] [0..2]	0, 0, 0	7, 145, 85	0	1602	18											
	1673	3	inter_mode_probs [2] [0..2]	0, 0, 0	7, 166, 63	0	1605	21											
	167	3	inter_mode_prob	0, 0, 0	7,	0	16	24											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	6		s [3] [0..2]		94, 66		08												
	1679	3	inter_mode_probs [4] [0..2]	0, 0, 0	8, 64, 46	0	1611	27											
	1682	3	inter_mode_probs [5] [0..2]	0, 0, 0	17, 81, 31	0	1614	30											
	1685	3	inter_mode_probs [6] [0..2]	0, 0, 0	25, 29, 30	0	1617	33											
	1688	2	switchable_interp_probs [0] [0..1]	0, 0	235, 162	0	1620	36			52-63								
	1690	2	switchable_interp_probs [1] [0..1]	0, 0	36, 255	0	1622	38											
	1692	2	switchable_interp_probs [2] [0..1]	0, 0	34, 3	0	1624	40											
	1694	2	switchable_interp_probs [3] [0..1]	0, 0	149, 14	0	1626	42											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					4													
	1696	4	intra_inter_probs [0..3]	0, 0, 0, 0	9, 102, 187, 225	0	16	28	44	64-71								
	1700	5	comp_inter_probs [0..4]	0, 0, 0, 0, 0	239, 183, 119, 96, 41	0	16	32	48	72-81								
	1705	2	single_ref_probs [0] [0..1]	0, 0	33, 16	0	16	37	53	82-101								
	1707	2	single_ref_probs [1] [0..1]	0, 0	77, 74	0	16	39	55									
	1709	2	single_ref_probs [2] [0..1]	0, 0	142, 142	0	16	41	57									
	171	2	single_ref_probs	0, 0	17	0	16	59										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1		[3] [0..1]		2, 170		43											
	1713	2	single_ref_probs [4] [0..1]	0, 0	238, 247	0	1645	61										
	1715	5	comp_ref_probs [0..4]	0, 0, 0, 0, 0	50, 126, 123, 221, 226	0	1647	63		102-111								
	1720	9	y_mode_probs [0] [0..8]	0, 0, 0, 0, 0, 0, 0, 0, 0	65, 32, 18, 144, 162, 194, 41, 51,	0	1652	68		112-151								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					98													
	1729	9	y_mode_probs [1] [0..8]	0, 0, 0, 0, 0, 0, 0, 0, 0	132, 68, 18, 165, 217, 196, 45, 40, 78	0	1661	77										
	1738	9	y_mode_probs [2] [0..8]	0, 0, 0, 0, 0, 0, 0, 0, 0	173, 80, 19, 176, 240, 193, 64, 35, 46	0	1670	86										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	1747	9	y_mode_probs [3] [0..8]	0, 0, 0, 0, 0, 0, 0, 0, 0	221, 135, 38, 194, 248, 121, 96, 85, 29	0	1679	95										
	1756	3	partition_probs [0] [0..2]	158, 97, 94	199, 122, 141	0	1688	104		152-215								
	1759	3	partition_probs [1] [0..2]	93, 24, 99	147, 63, 159	0	1691	107										
	1762	3	partition_probs [2] [0..2]	85, 119, 44	148,	0	1694	110										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					133, 118													
	1765	3	partition_probs [3] [0..2]	62, 59, 67	121, 104, 114	0	1697	113										
	1768	3	partition_probs [4] [0..2]	149, 53, 53	174, 73, 87	0	1700	116										
	1771	3	partition_probs [5] [0..2]	94, 20, 48	92, 41, 83	0	1703	119										
	1774	3	partition_probs [6] [0..2]	83, 53, 24	82, 99, 50	0	1706	122										
	1777	3	partition_probs [7] [0..2]	52, 18, 18	53, 39, 39	0	1709	125										
	1780	3	partition_probs [8] [0..2]	150, 40, 39	177, 58,	0	1712	128										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					59														
	1783	3	partition_probs [9] [0..2]	78, 12, 26	68, 26, 63	0	1715	131											
	1786	3	partition_probs [10] [0..2]	67, 33, 11	52, 79, 25	0	1718	134											
	1789	3	partition_probs [11] [0..2]	24, 7, 5	17, 14, 12	0	1721	137											
	1792	3	partition_probs [12] [0..2]	174, 35, 49	222, 34, 30	0	1724	140											
	1795	3	partition_probs [13] [0..2]	68, 11, 27	72, 16, 44	0	1727	143											
	1798	3	partition_probs [14] [0..2]	57, 15, 9	58, 32, 12	0	1730	146											
	1801	3	partition_probs [15] [0..2]	12, 3, 3	10, 7, 6	0	1733	149											
	1804	3	mvc_joints [3]	?,?,?	?,?,?	0	1736	152		MV COUNTERS	216-219								
	180	1	mv_sign	0	12	0	17	15			220-								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	7		[0]		8		39	5		221								
	1808	10	mv_classes [0] [0..9]	0, 0, 0, 0, 0, 0, 0, 0, 0, 0	224, 144, 192, 168, 192, 176, 192, 198, 198, 245	0	1740	156		222-232								
	1818	1	mv_class0 [0] [0..0]	0	216	0	1750	166		233-234								
	1819	10	mv_bits [0] [0..9]	0, 0, 0, 0, 0, 0, 0, 0, 0, 0	136, 140,	0	1751	167		235-254								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					148, 160, 176, 192, 224, 234, 234, 240														
	1829	1	mv_sign [1]	0	128	0	1761	177			255-256								
	1830	10	mv_classes [1] [0..9]	0, 0, 0, 0, 0, 0, 0, 0, 0, 0	216, 128, 176, 160, 176,	0	1762	178			257-267								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					176, 192, 198, 198, 208														
	1840	1	mv_class0 [1] [0..0]	0	208	0	1772	188			268-269								
	1841	10	mv_bits [1] [0..9]	0, 0, 0, 0, 0, 0, 0, 0, 0, 0	136, 140, 148, 160, 176, 192, 224, 234,	0	1773	189			270-289								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					234, 240													
	1851	3	mv_class0_fp [0] [0] [0..2]	0, 0, 0	128, 128, 64	0	1783	199		290-297								
	1854	3	mv_class0_fp [0] [1] [0..2]	0, 0, 0	96, 112, 64	0	1786	202										
	1863	3	mv_fp [0] [0..2]	0, 0, 0	64, 96, 64	0	1789	205		298-301								
	1857	3	mv_class0_fp [1] [0] [0..2]	0, 0, 0	128, 128, 64	0	1792	208		302-309								
	1860	3	mv_class0_fp [1] [1] [0..2]	0, 0, 0	96, 112, 64	0	1795	211										
	186	3	mv_fp	0, 0, 0	64,	0	17	21		310-								

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	6		[1] [0..2]		96, 64		98	4		313								
	1869	2	mv_class0_hp [0..1]	0, 0	160, 160	0	1801	217		314-315								
	1871	2	mv_hp [0..1]	0, 0	128, 128	0	1803	219		316-317								
	1873	47	DUMMY	0, 0, 0,

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults	Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
								4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					0, 0														
CL aligned	1920	9	uv_mode_probs [0] [0..8]	144, 11, 54, 157, 195, 130, 46, 58, 108	120, 7, 76, 176, 208, 126, 28, 54, 103	0	1805	221	240	MODE COUNTERS (Others)	318-417								
	1929	9	uv_mode_probs [1] [0..8]	118, 15, 123, 148, 131, 101, 44, 93, 131	48, 12, 154, 155, 139, 90, 34, 117,	0	1814	230											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					119													
	1938	9	uv_mode_probs [2] [0..8]	113, 12, 23, 188, 226, 142, 26, 32, 125	67, 6, 25, 204, 243, 158, 13, 21, 96	0	1823	239										
	1947	9	uv_mode_probs [3] [0..8]	120, 11, 50, 123, 163, 135, 64, 77, 103	97, 5, 44, 131, 176, 139, 48, 68, 97	0	1832	248										
	195	9	uv_mode_probs	113, 9,	83,	0	18	25										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
	6		[4] [0..8]	36, 155, 111, 157, 32, 44, 161	5, 42, 156, 111, 152, 26, 49, 152		41	7										
	1965	9	uv_mode_probs [5] [0..8]	116, 9, 55, 176, 76, 96, 37, 61, 149	80, 5, 58, 178, 74, 83, 33, 62, 145	0	1850	266										
	1974	9	uv_mode_probs [6] [0..8]	115, 9, 28, 141, 161, 167,	86, 5, 32, 154,	0	1859	275										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
											4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
				21, 25, 193	192, 168, 14, 22, 163													
	1983	9	uv_mode_probs [7] [0..8]	120, 12, 32, 145, 195, 142, 32, 38, 86	85, 5, 32, 156, 216, 148, 19, 29, 73	0	1868	284										
	1992	9	uv_mode_probs [8] [0..8]	116, 12, 64, 120, 140, 125, 49, 115, 121	77, 7, 64, 116, 132, 12	0	1877	293										

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					2, 37, 126, 120														
	2001	9	uv_mode_probs [9] [0..8]	102, 19, 66, 162, 182, 122, 35, 59, 128	101, 21, 107, 181, 192, 103, 19, 67, 125	0	1886	302											
	2010	7	seg_tree_probs [0..6]	255, 255, 255, 255, 255, 255, 255	255, 255, 255, 255, 25	0	1895	311											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					5, 255, 255, 255														
	2017	3	seg_pred_probs [0..2]	255, 255, 255	255, 255, 255	0	1902	318											
	2020	28	DUMMY	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0	115	1905	321											

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults				Capture At DV_CNT		State counter EBB Address	Coefficient counter EBB Address								
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
					0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0															
CL aligned	2048																		

Stream-in formats for creating compressed header

The following memory surfaces are input to PAK for Compressed Header coding

i. Prob Diff Surface

In Probability Diff Surface, there are 1805 8-bit Probability Diffs. Each of them corresponding to a Probability Diff in Compressed Header syntax. Although, for a given compressed header, not all the Probability Diff would be coded (depends on update flag), Probability Diff Surface is fully populated with 1805 entries (1805*8 / 512 = 29 cachelines). The 1805 8-bit Probability Diffs are expected to follow Compressed Header syntax order and fully packed.

ii. Compressed Header Syntax Surface

Each of the Compressed header Coding element (described in (2)) is represented by a 4-bit field. These 4-bit fields follows Compressed Header Syntax. Each of the field has a valid, Bin_probDiff_select, Prob_select, Bin as described in the table below.

	Description
Valid	Set to 1 if this is a valid Bin OR ProbabilityDiff field to code; Set to 0 to skip coding this field
Bin_ProbDiff_select	Set to 1 if Current field is a Bin (corresponding Prob, Bin are indicated by next 2 bits); Set to 0 if Current field is Probability Diff (probability diff to be coded is located in probability surface - ReMap)
Prob_Select	If current field is Bin, set to 1 if prob is 252; set to 0 if prob is 128
Bin	if current field is Bin, this is Bin value to be encoded

Compressed Header Syntax Surface is a fixed length surface. For syntax that should not be coded, valid bit should be set to 0. Total length of Compressed Header syntax Surface has 4033 Coding elements (16132 bits in 32 cachelines):

1805 Prob Diff and Prob Update flag

4 is_coeff_updated flag (per 4x4, 8x8, 16x16, 32x32)

5 control fields (MIN (tx_mode, ALLOW_32x32), tx_mode == TX_MODE_SELECT, use_compound_pred, use_hybrid_pred)

Probability Statistics Counters Table

In VP9 encoding, statistic collection is required to utilize the Forward Adaptation feature, which can substantially improve decoding performance because no intermediate computations based on encountered tokens is necessary. After encoding a frame, the VP9 encoder writes a Statistics Counter in the memory surface (HCP_IND_OBJ_BASE_ADDR_STATE: HCP VP9 PAK Probability Counter Streamout). The counter written in the memory surface is the same order as the VP9 Compressed header format.

The Statistics Counters collect frame statistics per symbol. In the VP9 Encoder, the PAK engine is responsible for statistics counter collection. The Kernel performs probability updates based on statistics counter values.

Each counter has has an associated context according to the VP9 Specification/Model.

Probability Statistics Counters Table

Note: The following table is arranged such that each row corresponds to sixteen 32-bit counters. In other words, each row corresponds to 512 bits (1 Cacheline). The VP9 Encoder PAK engine will have a total of 193 cacheline writers in the memory surface (HCP_IND_OBJ_BASW_ADDR_STATE: HCP VP9 PAK Probability Counter Streamout) at the end of each frame.

For brevity, data in the table is abbreviated. A Context Legend table has been provided after the Probability Statistics Counters table that explains the data in more detail.

R*xxx* Value 1 row = 1 Cacheline	**16 32-bit counters**															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
R0	ct_tx_8x8				ct_tx_16x16						ct_tx_32x32					
	[0] [0]	[0] [1]	[1] [0]	[1] [1]	[0] [0]	[0] [1]	[0] [2]	[1] [0]	[1] [1]	[1] [2]	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[1] [0]	[1] [1]
R1	ct_tx_32x32		ct_coef													
	[1] [2]	[1] [3]	[T=0] [B=0] [R=0] [CB=0] [CT=0] [N=0]	[T=0] [B=0] [R=0] [CB=0] [CT=0] [N=1]	[T=0] [B=0] [R=0] [CB=0] [CT=0] [N=2]	[T=0] [B=0] [R=0] [CB=0] [CT=0] [N=3]	[T=0] [B=0] [R=0] [CB=0] [CT=1] [N=0]	[T=0] [B=0] [R=0] [CB=0] [CT=1] [N=1]	[T=0] [B=0] [R=0] [CB=0] [CT=1] [N=2]	[T=0] [B=0] [R=0] [CB=0] [CT=1] [N=3]	[T=0] [B=0] [R=0] [CB=0] [CT=2] [N=0]	[T=0] [B=0] [R=0] [CB=0] [CT=2] [N=1]	[T=0] [B=0] [R=0] [CB=0] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=0] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=0] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=0] [N=1]
R2	ct_coef															
	[T=0] [B=0] [R=0] [CB=1] [CT=0] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=0] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=1] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=1] [N=1]	[T=0] [B=0] [R=0] [CB=1] [CT=1] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=1] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=2] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=2] [N=1]	[T=0] [B=0] [R=0] [CB=1] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=3] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=3] [N=1]	[T=0] [B=0] [R=0] [CB=1] [CT=3] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=3] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=4] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=4] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R3	ct_coef															
	[T=0] [B=0] [R=0] [CB=1] [CT=4] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=4] [N=3]	[T=0] [B=0] [R=0] [CB=1] [CT=5] [N=0]	[T=0] [B=0] [R=0] [CB=1] [CT=5] [N=1]	[T=0] [B=0] [R=0] [CB=1] [CT=5] [N=2]	[T=0] [B=0] [R=0] [CB=1] [CT=5] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=0] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=0] [N=1]	[T=0] [B=0] [R=0] [CB=2] [CT=0] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=0] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=1] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=1] [N=1]	[T=0] [B=0] [R=0] [CB=2] [CT=1] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=1] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=2] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=2] [N=1]
R4	ct_coef															
	[T=0] [B=0] [R=0] [CB=2] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=3] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=3] [N=1]	[T=0] [B=0] [R=0] [CB=2] [CT=3] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=3] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=4] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=4] [N=1]	[T=0] [B=0] [R=0] [CB=2] [CT=4] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=4] [N=3]	[T=0] [B=0] [R=0] [CB=2] [CT=5] [N=0]	[T=0] [B=0] [R=0] [CB=2] [CT=5] [N=1]	[T=0] [B=0] [R=0] [CB=2] [CT=5] [N=2]	[T=0] [B=0] [R=0] [CB=2] [CT=5] [N=3]	[T=0] [B=0] [R=0] [CB=3] [CT=0] [N=0]	[T=0] [B=0] [R=0] [CB=3] [CT=0] [N=1]
R5	ct_coef															
	[T=0] [B=0] [R=0] [CB=3] [CT=0] [N=2]	[T=0] [B=0] [R=0] [CB=3] [CT=0] [N=3]	[T=0] [B=0] [R=0] [CB=3] [CT=1] [N=0]	[T=0] [B=0] [R=0] [CB=3] [CT=1] [N=1]	[T=0] [B=0] [R=0] [CB=3] [CT=1] [N=2]	[T=0] [B=0] [R=0] [CB=3] [CT=1] [N=3]	[T=0] [B=0] [R=0] [CB=3] [CT=2] [N=0]	[T=0] [B=0] [R=0] [CB=3] [CT=2] [N=1]	[T=0] [B=0] [R=0] [CB=3] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=3] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=3] [CT=3] [N=0]	[T=0] [B=0] [R=0] [CB=3] [CT=3] [N=1]	[T=0] [B=0] [R=0] [CB=3] [CT=3] [N=2]	[T=0] [B=0] [R=0] [CB=3] [CT=3] [N=3]	[T=0] [B=0] [R=0] [CB=3] [CT=4] [N=0]	[T=0] [B=0] [R=0] [CB=3] [CT=4] [N=1]
R6	ct_coef															
	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]	[T=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=0] [R=0] [CB=3] [CT=4] [N=2]	[B=0] [R=0] [CB=3] [CT=4] [N=3]	[B=0] [R=0] [CB=3] [CT=5] [N=0]	[B=0] [R=0] [CB=3] [CT=5] [N=1]	[B=0] [R=0] [CB=3] [CT=5] [N=2]	[B=0] [R=0] [CB=3] [CT=5] [N=3]	[B=0] [R=0] [CB=4] [CT=0] [N=0]	[B=0] [R=0] [CB=4] [CT=0] [N=1]	[B=0] [R=0] [CB=4] [CT=0] [N=2]	[B=0] [R=0] [CB=4] [CT=0] [N=3]	[B=0] [R=0] [CB=4] [CT=1] [N=0]	[B=0] [R=0] [CB=4] [CT=1] [N=1]	[B=0] [R=0] [CB=4] [CT=1] [N=2]	[B=0] [R=0] [CB=4] [CT=1] [N=3]	[B=0] [R=0] [CB=4] [CT=2] [N=0]	[B=0] [R=0] [CB=4] [CT=2] [N=1]
R7	ct_coef															
	[T=0] [B=0] [R=0] [CB=4] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=4] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=4] [CT=3] [N=0]	[T=0] [B=0] [R=0] [CB=4] [CT=3] [N=1]	[T=0] [B=0] [R=0] [CB=4] [CT=3] [N=2]	[T=0] [B=0] [R=0] [CB=4] [CT=3] [N=3]	[T=0] [B=0] [R=0] [CB=4] [CT=4] [N=0]	[T=0] [B=0] [R=0] [CB=4] [CT=4] [N=1]	[T=0] [B=0] [R=0] [CB=4] [CT=4] [N=2]	[T=0] [B=0] [R=0] [CB=4] [CT=4] [N=3]	[T=0] [B=0] [R=0] [CB=4] [CT=5] [N=0]	[T=0] [B=0] [R=0] [CB=4] [CT=5] [N=1]	[T=0] [B=0] [R=0] [CB=4] [CT=5] [N=2]	[T=0] [B=0] [R=0] [CB=4] [CT=5] [N=3]	[T=0] [B=0] [R=0] [CB=5] [CT=0] [N=0]	[T=0] [B=0] [R=0] [CB=5] [CT=0] [N=1]
R8	ct_coef															
	[T=0] [B=0] [R=0] [CB=5] [CT=0] [N=2]	[T=0] [B=0] [R=0] [CB=5] [CT=0] [N=3]	[T=0] [B=0] [R=0] [CB=5] [CT=1] [N=0]	[T=0] [B=0] [R=0] [CB=5] [CT=1] [N=1]	[T=0] [B=0] [R=0] [CB=5] [CT=1] [N=2]	[T=0] [B=0] [R=0] [CB=5] [CT=1] [N=3]	[T=0] [B=0] [R=0] [CB=5] [CT=2] [N=0]	[T=0] [B=0] [R=0] [CB=5] [CT=2] [N=1]	[T=0] [B=0] [R=0] [CB=5] [CT=2] [N=2]	[T=0] [B=0] [R=0] [CB=5] [CT=2] [N=3]	[T=0] [B=0] [R=0] [CB=5] [CT=3] [N=0]	[T=0] [B=0] [R=0] [CB=5] [CT=3] [N=1]	[T=0] [B=0] [R=0] [CB=5] [CT=3] [N=2]	[T=0] [B=0] [R=0] [CB=5] [CT=3] [N=3]	[T=0] [B=0] [R=0] [CB=5] [CT=4] [N=0]	[T=0] [B=0] [R=0] [CB=5] [CT=4] [N=1]
R9	ct_coef															
	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=0]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]	[T=0] [B=0] [R=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=5] [CT=4] [N=2]	[CB=5] [CT=4] [N=3]	[CB=5] [CT=5] [N=0]	[CB=5] [CT=5] [N=1]	[CB=5] [CT=5] [N=2]	[CB=5] [CT=5] [N=3]	[CB=0] [CT=0] [N=0]	[CB=0] [CT=0] [N=1]	[CB=0] [CT=0] [N=2]	[CB=0] [CT=0] [N=3]	[CB=0] [CT=1] [N=0]	[CB=0] [CT=1] [N=1]	[CB=0] [CT=1] [N=2]	[CB=0] [CT=1] [N=3]	[CB=0] [CT=2] [N=0]	[CB=0] [CT=2] [N=1]
R10	ct_coef															
	[T=0] [B=0] [R=1] [CB=0] [CT=2] [N=2]	[T=0] [B=0] [R=1] [CB=0] [CT=2] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=0] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=0] [N=1]	[T=0] [B=0] [R=1] [CB=1] [CT=0] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=0] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=1] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=1] [N=1]	[T=0] [B=0] [R=1] [CB=1] [CT=1] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=1] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=2] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=2] [N=1]	[T=0] [B=0] [R=1] [CB=1] [CT=2] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=2] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=3] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=3] [N=1]
R11	ct_coef															
	[T=0] [B=0] [R=1] [CB=1] [CT=3] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=3] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=4] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=4] [N=1]	[T=0] [B=0] [R=1] [CB=1] [CT=4] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=4] [N=3]	[T=0] [B=0] [R=1] [CB=1] [CT=5] [N=0]	[T=0] [B=0] [R=1] [CB=1] [CT=5] [N=1]	[T=0] [B=0] [R=1] [CB=1] [CT=5] [N=2]	[T=0] [B=0] [R=1] [CB=1] [CT=5] [N=3]	[T=0] [B=0] [R=1] [CB=2] [CT=0] [N=0]	[T=0] [B=0] [R=1] [CB=2] [CT=0] [N=1]	[T=0] [B=0] [R=1] [CB=2] [CT=0] [N=2]	[T=0] [B=0] [R=1] [CB=2] [CT=0] [N=3]	[T=0] [B=0] [R=1] [CB=2] [CT=1] [N=0]	[T=0] [B=0] [R=1] [CB=2] [CT=1] [N=1]
R12	ct_coef															
	[T=0] [B=0] [R=1] [CB=2] [CT=1]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=2] [CT=5]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[N=2]	[CT=1] [N=3]	[CT=2] [N=0]	[CT=2] [N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[CT=3] [N=0]	[CT=3] [N=1]	[CT=3] [N=2]	[CT=3] [N=3]	[CT=4] [N=0]	[CT=4] [N=1]	[CT=4] [N=2]	[CT=4] [N=3]	[CT=5] [N=0]	[N=1]
R13	ct_coef															
	[T=0] [B=0] [R=1] [CB=2] [CT=5] [N=2]	[T=0] [B=0] [R=1] [CB=2] [CT=5] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=0] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=0] [N=1]	[T=0] [B=0] [R=1] [CB=3] [CT=0] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=0] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=1] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=1] [N=1]	[T=0] [B=0] [R=1] [CB=3] [CT=1] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=1] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=2] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=2] [N=1]	[T=0] [B=0] [R=1] [CB=3] [CT=2] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=2] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=3] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=3] [N=1]
R14	ct_coef															
	[T=0] [B=0] [R=1] [CB=3] [CT=3] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=3] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=4] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=4] [N=1]	[T=0] [B=0] [R=1] [CB=3] [CT=4] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=4] [N=3]	[T=0] [B=0] [R=1] [CB=3] [CT=5] [N=0]	[T=0] [B=0] [R=1] [CB=3] [CT=5] [N=1]	[T=0] [B=0] [R=1] [CB=3] [CT=5] [N=2]	[T=0] [B=0] [R=1] [CB=3] [CT=5] [N=3]	[T=0] [B=0] [R=1] [CB=4] [CT=0] [N=0]	[T=0] [B=0] [R=1] [CB=4] [CT=0] [N=1]	[T=0] [B=0] [R=1] [CB=4] [CT=0] [N=2]	[T=0] [B=0] [R=1] [CB=4] [CT=0] [N=3]	[T=0] [B=0] [R=1] [CB=4] [CT=1] [N=0]	[T=0] [B=0] [R=1] [CB=4] [CT=1] [N=1]
R15	ct_coef															
	[T=0] [B=0] [R=1] [CB=4] [CT=1] [N=2]	[T=0] [B=0] [R=1] [CB=4] [CT=1]	[T=0] [B=0] [R=1] [CB=4] [CT=2]	[T=0] [B=0] [R=1] [CB=4] [CT=2]	[T=0] [B=0] [R=1] [CB=4] [CT=2]	[T=0] [B=0] [R=1] [CB=4] [CT=2]	[T=0] [B=0] [R=1] [CB=4] [CT=3]	[T=0] [B=0] [R=1] [CB=4] [CT=3]	[T=0] [B=0] [R=1] [CB=4] [CT=3]	[T=0] [B=0] [R=1] [CB=4] [CT=3]	[T=0] [B=0] [R=1] [CB=4] [CT=4] [N=0]	[T=0] [B=0] [R=1] [CB=4] [CT=4] [N=1]	[T=0] [B=0] [R=1] [CB=4] [CT=4]	[T=0] [B=0] [R=1] [CB=4] [CT=4]	[T=0] [B=0] [R=1] [CB=4] [CT=5]	[T=0] [B=0] [R=1] [CB=4] [CT=5] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[N=3]	[N=0]	[N=1]	[N=2]	[N=3]	[N=0]	[N=1]	[N=2]	[N=3]			[N=2]	[N=3]	[N=0]	
R16	ct_coef															
	[T=0] [B=0] [R=1] [CB=4] [CT=5] [N=2]	[T=0] [B=0] [R=1] [CB=4] [CT=5] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=0] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=0] [N=1]	[T=0] [B=0] [R=1] [CB=5] [CT=0] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=0] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=1] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=1] [N=1]	[T=0] [B=0] [R=1] [CB=5] [CT=1] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=1] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=2] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=2] [N=1]	[T=0] [B=0] [R=1] [CB=5] [CT=2] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=2] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=3] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=3] [N=1]
R17	ct_coef															
	[T=0] [B=0] [R=1] [CB=5] [CT=3] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=3] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=4] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=4] [N=1]	[T=0] [B=0] [R=1] [CB=5] [CT=4] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=4] [N=3]	[T=0] [B=0] [R=1] [CB=5] [CT=5] [N=0]	[T=0] [B=0] [R=1] [CB=5] [CT=5] [N=1]	[T=0] [B=0] [R=1] [CB=5] [CT=5] [N=2]	[T=0] [B=0] [R=1] [CB=5] [CT=5] [N=3]	[T=0] [B=1] [R=0] [CB=0] [CT=0] [N=0]	[T=0] [B=1] [R=0] [CB=0] [CT=0] [N=1]	[T=0] [B=1] [R=0] [CB=0] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=0] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=0] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=0] [CT=1] [N=1]
R18	ct_coef															
	[T=0] [B=1] [R=0] [CB=0] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=0] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=0] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=0] [CT=2] [N=1]	[T=0] [B=1] [R=0] [CB=0] [CT=2] [N=2]	[T=0] [B=1] [R=0] [CB=0] [CT=2] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=0] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=0] [N=1]	[T=0] [B=1] [R=0] [CB=1] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=1] [N=1]	[T=0] [B=1] [R=0] [CB=1] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=2] [N=1]
R19	ct_coef															

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[T=0] [B=1] [R=0] [CB=1] [CT=2] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=2] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=3] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=3] [N=1]	[T=0] [B=1] [R=0] [CB=1] [CT=3] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=3] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=4] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=4] [N=1]	[T=0] [B=1] [R=0] [CB=1] [CT=4] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=4] [N=3]	[T=0] [B=1] [R=0] [CB=1] [CT=5] [N=0]	[T=0] [B=1] [R=0] [CB=1] [CT=5] [N=1]	[T=0] [B=1] [R=0] [CB=1] [CT=5] [N=2]	[T=0] [B=1] [R=0] [CB=1] [CT=5] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=0] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=0] [N=1]
R20	ct_coef															
	[T=0] [B=1] [R=0] [CB=2] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=1] [N=1]	[T=0] [B=1] [R=0] [CB=2] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=2] [N=1]	[T=0] [B=1] [R=0] [CB=2] [CT=2] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=2] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=3] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=3] [N=1]	[T=0] [B=1] [R=0] [CB=2] [CT=3] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=3] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=4] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=4] [N=1]
R21	ct_coef															
	[T=0] [B=1] [R=0] [CB=2] [CT=4] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=4] [N=3]	[T=0] [B=1] [R=0] [CB=2] [CT=5] [N=0]	[T=0] [B=1] [R=0] [CB=2] [CT=5] [N=1]	[T=0] [B=1] [R=0] [CB=2] [CT=5] [N=2]	[T=0] [B=1] [R=0] [CB=2] [CT=5] [N=3]	[T=0] [B=1] [R=0] [CB=3] [CT=0] [N=0]	[T=0] [B=1] [R=0] [CB=3] [CT=0] [N=1]	[T=0] [B=1] [R=0] [CB=3] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=3] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=3] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=3] [CT=1] [N=1]	[T=0] [B=1] [R=0] [CB=3] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=3] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=3] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=3] [CT=2] [N=1]
R22	ct_coef															
	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[R=0] [CB=3] [CT=2] [N=2]	[R=0] [CB=3] [CT=2] [N=3]	[R=0] [CB=3] [CT=3] [N=0]	[R=0] [CB=3] [CT=3] [N=1]	[R=0] [CB=3] [CT=3] [N=2]	[R=0] [CB=3] [CT=3] [N=3]	[R=0] [CB=3] [CT=4] [N=0]	[R=0] [CB=3] [CT=4] [N=1]	[R=0] [CB=3] [CT=4] [N=2]	[R=0] [CB=3] [CT=4] [N=3]	[R=0] [CB=3] [CT=5] [N=0]	[R=0] [CB=3] [CT=5] [N=1]	[R=0] [CB=3] [CT=5] [N=2]	[R=0] [CB=3] [CT=5] [N=3]	[R=0] [CB=4] [CT=0] [N=0]	[R=0] [CB=4] [CT=0] [N=1]
R23	ct_coef															
	[T=0] [B=1] [R=0] [CB=4] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=4] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=4] [CT=1] [N=1]	[T=0] [B=1] [R=0] [CB=4] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=4] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=4] [CT=2] [N=1]	[T=0] [B=1] [R=0] [CB=4] [CT=2] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=2] [N=3]	[T=0] [B=1] [R=0] [CB=4] [CT=3] [N=0]	[T=0] [B=1] [R=0] [CB=4] [CT=3] [N=1]	[T=0] [B=1] [R=0] [CB=4] [CT=3] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=3] [N=3]	[T=0] [B=1] [R=0] [CB=4] [CT=4] [N=0]	[T=0] [B=1] [R=0] [CB=4] [CT=4] [N=1]
R24	ct_coef															
	[T=0] [B=1] [R=0] [CB=4] [CT=4] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=4] [N=3]	[T=0] [B=1] [R=0] [CB=4] [CT=5] [N=0]	[T=0] [B=1] [R=0] [CB=4] [CT=5] [N=1]	[T=0] [B=1] [R=0] [CB=4] [CT=5] [N=2]	[T=0] [B=1] [R=0] [CB=4] [CT=5] [N=3]	[T=0] [B=1] [R=0] [CB=5] [CT=0] [N=0]	[T=0] [B=1] [R=0] [CB=5] [CT=0] [N=1]	[T=0] [B=1] [R=0] [CB=5] [CT=0] [N=2]	[T=0] [B=1] [R=0] [CB=5] [CT=0] [N=3]	[T=0] [B=1] [R=0] [CB=5] [CT=1] [N=0]	[T=0] [B=1] [R=0] [CB=5] [CT=1] [N=1]	[T=0] [B=1] [R=0] [CB=5] [CT=1] [N=2]	[T=0] [B=1] [R=0] [CB=5] [CT=1] [N=3]	[T=0] [B=1] [R=0] [CB=5] [CT=2] [N=0]	[T=0] [B=1] [R=0] [CB=5] [CT=2] [N=1]
R25	ct_coef															
	[T=0] [B=1] [R=0] [CB=5]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=0]	[T=0] [B=1] [R=1]	[T=0] [B=1] [R=1] [CB=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CT=2] [N=2]	[CB=5] [CT=2] [N=3]	[CB=5] [CT=3] [N=0]	[CB=5] [CT=3] [N=1]	[CB=5] [CT=3] [N=2]	[CB=5] [CT=3] [N=3]	[CB=5] [CT=4] [N=0]	[CB=5] [CT=4] [N=1]	[CB=5] [CT=4] [N=2]	[CB=5] [CT=4] [N=3]	[CB=5] [CT=5] [N=0]	[CB=5] [CT=5] [N=1]	[CB=5] [CT=5] [N=2]	[CB=5] [CT=5] [N=3]	[CB=0] [CT=0] [N=0]	[CT=0] [N=1]
R26	ct_coef															
	[T=0] [B=1] [R=1] [CB=0] [CT=0] [N=2]	[T=0] [B=1] [R=1] [CB=0] [CT=0] [N=3]	[T=0] [B=1] [R=1] [CB=0] [CT=1] [N=0]	[T=0] [B=1] [R=1] [CB=0] [CT=1] [N=1]	[T=0] [B=1] [R=1] [CB=0] [CT=1] [N=2]	[T=0] [B=1] [R=1] [CB=0] [CT=1] [N=3]	[T=0] [B=1] [R=1] [CB=0] [CT=2] [N=0]	[T=0] [B=1] [R=1] [CB=0] [CT=2] [N=1]	[T=0] [B=1] [R=1] [CB=0] [CT=2] [N=2]	[T=0] [B=1] [R=1] [CB=0] [CT=2] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=0] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=0] [N=1]	[T=0] [B=1] [R=1] [CB=1] [CT=0] [N=2]	[T=0] [B=1] [R=1] [CB=1] [CT=0] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=1] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=1] [N=1]
R27	ct_coef															
	[T=0] [B=1] [R=1] [CB=1] [CT=1] [N=2]	[T=0] [B=1] [R=1] [CB=1] [CT=1] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=2] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=2] [N=1]	[T=0] [B=1] [R=1] [CB=1] [CT=2] [N=2]	[T=0] [B=1] [R=1] [CB=1] [CT=2] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=3] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=3] [N=1]	[T=0] [B=1] [R=1] [CB=1] [CT=3] [N=2]	[T=0] [B=1] [R=1] [CB=1] [CT=3] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=4] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=4] [N=1]	[T=0] [B=1] [R=1] [CB=1] [CT=4] [N=2]	[T=0] [B=1] [R=1] [CB=1] [CT=4] [N=3]	[T=0] [B=1] [R=1] [CB=1] [CT=5] [N=0]	[T=0] [B=1] [R=1] [CB=1] [CT=5] [N=1]
R28	ct_coef															
	[T=0] [B=1] [R=1] [CB=1] [CT=5] [N=2]	[T=0] [B=1] [R=1] [CB=1]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2] [CT=2]	[T=0] [B=1] [R=1] [CB=2] [CT=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2]	[T=0] [B=1] [R=1] [CB=2] [CT=3] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=5] [N=3]	[CT=0] [N=0]	[CT=0] [N=1]	[CT=0] [N=2]	[CT=0] [N=3]	[CT=1] [N=0]	[CT=1] [N=1]	[CT=1] [N=2]	[CT=1] [N=3]	[N=0]	[N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[CT=3] [N=0]	
R29	ct_coef															
	[T=0] [B=1] [R=1] [CB=2] [CT=3] [N=2]	[T=0] [B=1] [R=1] [CB=2] [CT=3] [N=3]	[T=0] [B=1] [R=1] [CB=2] [CT=4] [N=0]	[T=0] [B=1] [R=1] [CB=2] [CT=4] [N=1]	[T=0] [B=1] [R=1] [CB=2] [CT=4] [N=2]	[T=0] [B=1] [R=1] [CB=2] [CT=4] [N=3]	[T=0] [B=1] [R=1] [CB=2] [CT=5] [N=0]	[T=0] [B=1] [R=1] [CB=2] [CT=5] [N=1]	[T=0] [B=1] [R=1] [CB=2] [CT=5] [N=2]	[T=0] [B=1] [R=1] [CB=2] [CT=5] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=0] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=0] [N=1]	[T=0] [B=1] [R=1] [CB=3] [CT=0] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=0] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=1] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=1] [N=1]
R30	ct_coef															
	[T=0] [B=1] [R=1] [CB=3] [CT=1] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=1] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=2] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=2] [N=1]	[T=0] [B=1] [R=1] [CB=3] [CT=2] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=2] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=3] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=3] [N=1]	[T=0] [B=1] [R=1] [CB=3] [CT=3] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=3] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=4] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=4] [N=1]	[T=0] [B=1] [R=1] [CB=3] [CT=4] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=4] [N=3]	[T=0] [B=1] [R=1] [CB=3] [CT=5] [N=0]	[T=0] [B=1] [R=1] [CB=3] [CT=5] [N=1]
R31	ct_coef															
	[T=0] [B=1] [R=1] [CB=3] [CT=5] [N=2]	[T=0] [B=1] [R=1] [CB=3] [CT=5] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=0] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=0] [N=1]	[T=0] [B=1] [R=1] [CB=4] [CT=0] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=0] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=1] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=1] [N=1]	[T=0] [B=1] [R=1] [CB=4] [CT=1] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=1] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=2] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=2] [N=1]	[T=0] [B=1] [R=1] [CB=4] [CT=2] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=2] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=3] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=3] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R32	ct_coef															
	[T=0] [B=1] [R=1] [CB=4] [CT=3] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=3] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=4] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=4] [N=1]	[T=0] [B=1] [R=1] [CB=4] [CT=4] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=4] [N=3]	[T=0] [B=1] [R=1] [CB=4] [CT=5] [N=0]	[T=0] [B=1] [R=1] [CB=4] [CT=5] [N=1]	[T=0] [B=1] [R=1] [CB=4] [CT=5] [N=2]	[T=0] [B=1] [R=1] [CB=4] [CT=5] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=0] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=0] [N=1]	[T=0] [B=1] [R=1] [CB=5] [CT=0] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=0] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=1] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=1] [N=1]
R33	ct_coef															
	[T=0] [B=1] [R=1] [CB=5] [CT=1] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=1] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=2] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=2] [N=1]	[T=0] [B=1] [R=1] [CB=5] [CT=2] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=2] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=3] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=3] [N=1]	[T=0] [B=1] [R=1] [CB=5] [CT=3] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=3] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=4] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=4] [N=1]	[T=0] [B=1] [R=1] [CB=5] [CT=4] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=4] [N=3]	[T=0] [B=1] [R=1] [CB=5] [CT=5] [N=0]	[T=0] [B=1] [R=1] [CB=5] [CT=5] [N=1]
R34	ct_coef															
	[T=0] [B=1] [R=1] [CB=5] [CT=5] [N=2]	[T=0] [B=1] [R=1] [CB=5] [CT=5] [N=3]	[T=1] [B=0] [R=0] [CB=0] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=0] [CT=0] [N=1]	[T=1] [B=0] [R=0] [CB=0] [CT=0] [N=2]	[T=1] [B=0] [R=0] [CB=0] [CT=0] [N=3]	[T=1] [B=0] [R=0] [CB=0] [CT=1] [N=0]	[T=1] [B=0] [R=0] [CB=0] [CT=1] [N=1]	[T=1] [B=0] [R=0] [CB=0] [CT=1] [N=2]	[T=1] [B=0] [R=0] [CB=0] [CT=1] [N=3]	[T=1] [B=0] [R=0] [CB=0] [CT=2] [N=0]	[T=1] [B=0] [R=0] [CB=0] [CT=2] [N=1]	[T=1] [B=0] [R=0] [CB=0] [CT=2] [N=2]	[T=1] [B=0] [R=0] [CB=0] [CT=2] [N=3]	[T=1] [B=0] [R=0] [CB=1] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=1] [CT=0] [N=1]
R35	ct_coef															
	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=0] [R=0] [CB=1] [CT=0] [N=2]	[B=0] [R=0] [CB=1] [CT=0] [N=3]	[B=0] [R=0] [CB=1] [CT=1] [N=0]	[B=0] [R=0] [CB=1] [CT=1] [N=1]	[B=0] [R=0] [CB=1] [CT=1] [N=2]	[B=0] [R=0] [CB=1] [CT=1] [N=3]	[B=0] [R=0] [CB=1] [CT=2] [N=0]	[B=0] [R=0] [CB=1] [CT=2] [N=1]	[B=0] [R=0] [CB=1] [CT=2] [N=2]	[B=0] [R=0] [CB=1] [CT=2] [N=3]	[B=0] [R=0] [CB=1] [CT=3] [N=0]	[B=0] [R=0] [CB=1] [CT=3] [N=1]	[B=0] [R=0] [CB=1] [CT=3] [N=2]	[B=0] [R=0] [CB=1] [CT=3] [N=3]	[B=0] [R=0] [CB=1] [CT=4] [N=0]	[B=0] [R=0] [CB=1] [CT=4] [N=1]
R36	ct_coef															
	[T=1] [B=0] [R=0] [CB=1] [CT=4] [N=2]	[T=1] [B=0] [R=0] [CB=1] [CT=4] [N=3]	[T=1] [B=0] [R=0] [CB=1] [CT=5] [N=0]	[T=1] [B=0] [R=0] [CB=1] [CT=5] [N=1]	[T=1] [B=0] [R=0] [CB=1] [CT=5] [N=2]	[T=1] [B=0] [R=0] [CB=1] [CT=5] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=0] [N=1]	[T=1] [B=0] [R=0] [CB=2] [CT=0] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=0] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=1] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=1] [N=1]	[T=1] [B=0] [R=0] [CB=2] [CT=1] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=1] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=2] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=2] [N=1]
R37	ct_coef															
	[T=1] [B=0] [R=0] [CB=2] [CT=2] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=2] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=3] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=3] [N=1]	[T=1] [B=0] [R=0] [CB=2] [CT=3] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=3] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=4] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=4] [N=1]	[T=1] [B=0] [R=0] [CB=2] [CT=4] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=4] [N=3]	[T=1] [B=0] [R=0] [CB=2] [CT=5] [N=0]	[T=1] [B=0] [R=0] [CB=2] [CT=5] [N=1]	[T=1] [B=0] [R=0] [CB=2] [CT=5] [N=2]	[T=1] [B=0] [R=0] [CB=2] [CT=5] [N=3]	[T=1] [B=0] [R=0] [CB=3] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=3] [CT=0] [N=1]
R38	ct_coef															
	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]	[T=1] [B=0] [R=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=3] [CT=0] [N=2]	[CB=3] [CT=0] [N=3]	[CB=3] [CT=1] [N=0]	[CB=3] [CT=1] [N=1]	[CB=3] [CT=1] [N=2]	[CB=3] [CT=1] [N=3]	[CB=3] [CT=2] [N=0]	[CB=3] [CT=2] [N=1]	[CB=3] [CT=2] [N=2]	[CB=3] [CT=2] [N=3]	[CB=3] [CT=3] [N=0]	[CB=3] [CT=3] [N=1]	[CB=3] [CT=3] [N=2]	[CB=3] [CT=3] [N=3]	[CB=3] [CT=4] [N=0]	[CB=3] [CT=4] [N=1]
R39	ct_coef															
	[T=1] [B=0] [R=0] [CB=3] [CT=4] [N=2]	[T=1] [B=0] [R=0] [CB=3] [CT=4] [N=3]	[T=1] [B=0] [R=0] [CB=3] [CT=5] [N=0]	[T=1] [B=0] [R=0] [CB=3] [CT=5] [N=1]	[T=1] [B=0] [R=0] [CB=3] [CT=5] [N=2]	[T=1] [B=0] [R=0] [CB=3] [CT=5] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=0] [N=1]	[T=1] [B=0] [R=0] [CB=4] [CT=0] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=0] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=1] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=1] [N=1]	[T=1] [B=0] [R=0] [CB=4] [CT=1] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=1] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=2] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=2] [N=1]
R40	ct_coef															
	[T=1] [B=0] [R=0] [CB=4] [CT=2] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=2] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=3] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=3] [N=1]	[T=1] [B=0] [R=0] [CB=4] [CT=3] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=3] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=4] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=4] [N=1]	[T=1] [B=0] [R=0] [CB=4] [CT=4] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=4] [N=3]	[T=1] [B=0] [R=0] [CB=4] [CT=5] [N=0]	[T=1] [B=0] [R=0] [CB=4] [CT=5] [N=1]	[T=1] [B=0] [R=0] [CB=4] [CT=5] [N=2]	[T=1] [B=0] [R=0] [CB=4] [CT=5] [N=3]	[T=1] [B=0] [R=0] [CB=5] [CT=0] [N=0]	[T=1] [B=0] [R=0] [CB=5] [CT=0] [N=1]
R41	ct_coef															
	[T=1] [B=0] [R=0] [CB=5] [CT=0]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5]	[T=1] [B=0] [R=0] [CB=5] [CT=4]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[N=2]	[CT=0] [N=3]	[CT=1] [N=0]	[CT=1] [N=1]	[CT=1] [N=2]	[CT=1] [N=3]	[CT=2] [N=0]	[CT=2] [N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[CT=3] [N=0]	[CT=3] [N=1]	[CT=3] [N=2]	[CT=3] [N=3]	[CT=4] [N=0]	[N=1]
R42	ct_coef															
	[T=1] [B=0] [R=0] [CB=5] [CT=4] [N=2]	[T=1] [B=0] [R=0] [CB=5] [CT=4] [N=3]	[T=1] [B=0] [R=0] [CB=5] [CT=5] [N=0]	[T=1] [B=0] [R=0] [CB=5] [CT=5] [N=1]	[T=1] [B=0] [R=0] [CB=5] [CT=5] [N=2]	[T=1] [B=0] [R=0] [CB=5] [CT=5] [N=3]	[T=1] [B=0] [R=1] [CB=0] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=0] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=0] [CT=0] [N=2]	[T=1] [B=0] [R=1] [CB=0] [CT=0] [N=3]	[T=1] [B=0] [R=1] [CB=0] [CT=1] [N=0]	[T=1] [B=0] [R=1] [CB=0] [CT=1] [N=1]	[T=1] [B=0] [R=1] [CB=0] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=0] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=0] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=0] [CT=2] [N=1]
R43	ct_coef															
	[T=1] [B=0] [R=1] [CB=0] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=0] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=1] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=1] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=1] [CT=0] [N=2]	[T=1] [B=0] [R=1] [CB=1] [CT=0] [N=3]	[T=1] [B=0] [R=1] [CB=1] [CT=1] [N=0]	[T=1] [B=0] [R=1] [CB=1] [CT=1] [N=1]	[T=1] [B=0] [R=1] [CB=1] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=1] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=1] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=1] [CT=2] [N=1]	[T=1] [B=0] [R=1] [CB=1] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=1] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=1] [CT=3] [N=0]	[T=1] [B=0] [R=1] [CB=1] [CT=3] [N=1]
R44	ct_coef															
	[T=1] [B=0] [R=1] [CB=1] [CT=3] [N=2]	[T=1] [B=0] [R=1] [CB=1] [CT=3]	[T=1] [B=0] [R=1] [CB=1] [CT=4]	[T=1] [B=0] [R=1] [CB=1] [CT=4]	[T=1] [B=0] [R=1] [CB=1] [CT=4]	[T=1] [B=0] [R=1] [CB=1] [CT=4]	[T=1] [B=0] [R=1] [CB=1] [CT=5]	[T=1] [B=0] [R=1] [CB=1] [CT=5]	[T=1] [B=0] [R=1] [CB=1] [CT=5]	[T=1] [B=0] [R=1] [CB=1] [CT=5]	[T=1] [B=0] [R=1] [CB=2] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=2] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=2] [CT=0]	[T=1] [B=0] [R=1] [CB=2] [CT=0]	[T=1] [B=0] [R=1] [CB=2] [CT=1]	[T=1] [B=0] [R=1] [CB=2] [CT=1] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[N=3]	[N=0]	[N=1]	[N=2]	[N=3]	[N=0]	[N=1]	[N=2]	[N=3]			[N=2]	[N=3]	[N=0]	
R45	ct_coef															
	[T=1] [B=0] [R=1] [CB=2] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=2] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=2] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=2] [CT=2] [N=1]	[T=1] [B=0] [R=1] [CB=2] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=2] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=2] [CT=3] [N=0]	[T=1] [B=0] [R=1] [CB=2] [CT=3] [N=1]	[T=1] [B=0] [R=1] [CB=2] [CT=3] [N=2]	[T=1] [B=0] [R=1] [CB=2] [CT=3] [N=3]	[T=1] [B=0] [R=1] [CB=2] [CT=4] [N=0]	[T=1] [B=0] [R=1] [CB=2] [CT=4] [N=1]	[T=1] [B=0] [R=1] [CB=2] [CT=4] [N=2]	[T=1] [B=0] [R=1] [CB=2] [CT=4] [N=3]	[T=1] [B=0] [R=1] [CB=2] [CT=5] [N=0]	[T=1] [B=0] [R=1] [CB=2] [CT=5] [N=1]
R46	ct_coef															
	[T=1] [B=0] [R=1] [CB=2] [CT=5] [N=2]	[T=1] [B=0] [R=1] [CB=2] [CT=5] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=3] [CT=0] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=0] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=1] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=1] [N=1]	[T=1] [B=0] [R=1] [CB=3] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=2] [N=1]	[T=1] [B=0] [R=1] [CB=3] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=3] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=3] [N=1]
R47	ct_coef															
	[T=1] [B=0] [R=1] [CB=3] [CT=3] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=3] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=4] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=4] [N=1]	[T=1] [B=0] [R=1] [CB=3] [CT=4] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=4] [N=3]	[T=1] [B=0] [R=1] [CB=3] [CT=5] [N=0]	[T=1] [B=0] [R=1] [CB=3] [CT=5] [N=1]	[T=1] [B=0] [R=1] [CB=3] [CT=5] [N=2]	[T=1] [B=0] [R=1] [CB=3] [CT=5] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=4] [CT=0] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=0] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=1] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=1] [N=1]
R48	ct_coef															

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[T=1] [B=0] [R=1] [CB=4] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=2] [N=1]	[T=1] [B=0] [R=1] [CB=4] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=3] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=3] [N=1]	[T=1] [B=0] [R=1] [CB=4] [CT=3] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=3] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=4] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=4] [N=1]	[T=1] [B=0] [R=1] [CB=4] [CT=4] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=4] [N=3]	[T=1] [B=0] [R=1] [CB=4] [CT=5] [N=0]	[T=1] [B=0] [R=1] [CB=4] [CT=5] [N=1]
R49	ct_coef															
	[T=1] [B=0] [R=1] [CB=4] [CT=5] [N=2]	[T=1] [B=0] [R=1] [CB=4] [CT=5] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=0] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=0] [N=1]	[T=1] [B=0] [R=1] [CB=5] [CT=0] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=0] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=1] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=1] [N=1]	[T=1] [B=0] [R=1] [CB=5] [CT=1] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=1] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=2] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=2] [N=1]	[T=1] [B=0] [R=1] [CB=5] [CT=2] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=2] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=3] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=3] [N=1]
R50	ct_coef															
	[T=1] [B=0] [R=1] [CB=5] [CT=3] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=3] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=4] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=4] [N=1]	[T=1] [B=0] [R=1] [CB=5] [CT=4] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=4] [N=3]	[T=1] [B=0] [R=1] [CB=5] [CT=5] [N=0]	[T=1] [B=0] [R=1] [CB=5] [CT=5] [N=1]	[T=1] [B=0] [R=1] [CB=5] [CT=5] [N=2]	[T=1] [B=0] [R=1] [CB=5] [CT=5] [N=3]	[T=1] [B=1] [R=0] [CB=0] [CT=0] [N=0]	[T=1] [B=1] [R=0] [CB=0] [CT=0] [N=1]	[T=1] [B=1] [R=0] [CB=0] [CT=0] [N=2]	[T=1] [B=1] [R=0] [CB=0] [CT=0] [N=3]	[T=1] [B=1] [R=0] [CB=0] [CT=1] [N=0]	[T=1] [B=1] [R=0] [CB=0] [CT=1] [N=1]
R51	ct_coef															
	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]	[T=1] [B=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[R=0] [CB=0] [CT=1] [N=2]	[R=0] [CB=0] [CT=1] [N=3]	[R=0] [CB=0] [CT=2] [N=0]	[R=0] [CB=0] [CT=2] [N=1]	[R=0] [CB=0] [CT=2] [N=2]	[R=0] [CB=0] [CT=2] [N=3]	[R=0] [CB=1] [CT=0] [N=0]	[R=0] [CB=1] [CT=0] [N=1]	[R=0] [CB=1] [CT=0] [N=2]	[R=0] [CB=1] [CT=0] [N=3]	[R=0] [CB=1] [CT=1] [N=0]	[R=0] [CB=1] [CT=1] [N=1]	[R=0] [CB=1] [CT=1] [N=2]	[R=0] [CB=1] [CT=1] [N=3]	[R=0] [CB=1] [CT=2] [N=0]	[R=0] [CB=1] [CT=2] [N=1]
R52	ct_coef															
	[T=1] [B=1] [R=0] [CB=1] [CT=2] [N=2]	[T=1] [B=1] [R=0] [CB=1] [CT=2] [N=3]	[T=1] [B=1] [R=0] [CB=1] [CT=3] [N=0]	[T=1] [B=1] [R=0] [CB=1] [CT=3] [N=1]	[T=1] [B=1] [R=0] [CB=1] [CT=3] [N=2]	[T=1] [B=1] [R=0] [CB=1] [CT=3] [N=3]	[T=1] [B=1] [R=0] [CB=1] [CT=4] [N=0]	[T=1] [B=1] [R=0] [CB=1] [CT=4] [N=1]	[T=1] [B=1] [R=0] [CB=1] [CT=4] [N=2]	[T=1] [B=1] [R=0] [CB=1] [CT=4] [N=3]	[T=1] [B=1] [R=0] [CB=1] [CT=5] [N=0]	[T=1] [B=1] [R=0] [CB=1] [CT=5] [N=1]	[T=1] [B=1] [R=0] [CB=1] [CT=5] [N=2]	[T=1] [B=1] [R=0] [CB=1] [CT=5] [N=3]	[T=1] [B=1] [R=0] [CB=2] [CT=0] [N=0]	[T=1] [B=1] [R=0] [CB=2] [CT=0] [N=1]
R53	ct_coef															
	[T=1] [B=1] [R=0] [CB=2] [CT=0] [N=2]	[T=1] [B=1] [R=0] [CB=2] [CT=0] [N=3]	[T=1] [B=1] [R=0] [CB=2] [CT=1] [N=0]	[T=1] [B=1] [R=0] [CB=2] [CT=1] [N=1]	[T=1] [B=1] [R=0] [CB=2] [CT=1] [N=2]	[T=1] [B=1] [R=0] [CB=2] [CT=1] [N=3]	[T=1] [B=1] [R=0] [CB=2] [CT=2] [N=0]	[T=1] [B=1] [R=0] [CB=2] [CT=2] [N=1]	[T=1] [B=1] [R=0] [CB=2] [CT=2] [N=2]	[T=1] [B=1] [R=0] [CB=2] [CT=2] [N=3]	[T=1] [B=1] [R=0] [CB=2] [CT=3] [N=0]	[T=1] [B=1] [R=0] [CB=2] [CT=3] [N=1]	[T=1] [B=1] [R=0] [CB=2] [CT=3] [N=2]	[T=1] [B=1] [R=0] [CB=2] [CT=3] [N=3]	[T=1] [B=1] [R=0] [CB=2] [CT=4] [N=0]	[T=1] [B=1] [R=0] [CB=2] [CT=4] [N=1]
R54	ct_coef															
	[T=1] [B=1] [R=0] [CB=2]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0] [CB=3]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CT=4] [N=2]	[CB=2] [CT=4] [N=3]	[CB=2] [CT=5] [N=0]	[CB=2] [CT=5] [N=1]	[CB=2] [CT=5] [N=2]	[CB=2] [CT=5] [N=3]	[CB=3] [CT=0] [N=0]	[CB=3] [CT=0] [N=1]	[CB=3] [CT=0] [N=2]	[CB=3] [CT=0] [N=3]	[CB=3] [CT=1] [N=0]	[CB=3] [CT=1] [N=1]	[CB=3] [CT=1] [N=2]	[CB=3] [CT=1] [N=3]	[CB=3] [CT=2] [N=0]	[CT=2] [N=1]
R55	ct_coef															
	[T=1] [B=1] [R=0] [CB=3] [CT=2] [N=2]	[T=1] [B=1] [R=0] [CB=3] [CT=2] [N=3]	[T=1] [B=1] [R=0] [CB=3] [CT=3] [N=0]	[T=1] [B=1] [R=0] [CB=3] [CT=3] [N=1]	[T=1] [B=1] [R=0] [CB=3] [CT=3] [N=2]	[T=1] [B=1] [R=0] [CB=3] [CT=3] [N=3]	[T=1] [B=1] [R=0] [CB=3] [CT=4] [N=0]	[T=1] [B=1] [R=0] [CB=3] [CT=4] [N=1]	[T=1] [B=1] [R=0] [CB=3] [CT=4] [N=2]	[T=1] [B=1] [R=0] [CB=3] [CT=4] [N=3]	[T=1] [B=1] [R=0] [CB=3] [CT=5] [N=0]	[T=1] [B=1] [R=0] [CB=3] [CT=5] [N=1]	[T=1] [B=1] [R=0] [CB=3] [CT=5] [N=2]	[T=1] [B=1] [R=0] [CB=3] [CT=5] [N=3]	[T=1] [B=1] [R=0] [CB=4] [CT=0] [N=0]	[T=1] [B=1] [R=0] [CB=4] [CT=0] [N=1]
R56	ct_coef															
	[T=1] [B=1] [R=0] [CB=4] [CT=0] [N=2]	[T=1] [B=1] [R=0] [CB=4] [CT=0] [N=3]	[T=1] [B=1] [R=0] [CB=4] [CT=1] [N=0]	[T=1] [B=1] [R=0] [CB=4] [CT=1] [N=1]	[T=1] [B=1] [R=0] [CB=4] [CT=1] [N=2]	[T=1] [B=1] [R=0] [CB=4] [CT=1] [N=3]	[T=1] [B=1] [R=0] [CB=4] [CT=2] [N=0]	[T=1] [B=1] [R=0] [CB=4] [CT=2] [N=1]	[T=1] [B=1] [R=0] [CB=4] [CT=2] [N=2]	[T=1] [B=1] [R=0] [CB=4] [CT=2] [N=3]	[T=1] [B=1] [R=0] [CB=4] [CT=3] [N=0]	[T=1] [B=1] [R=0] [CB=4] [CT=3] [N=1]	[T=1] [B=1] [R=0] [CB=4] [CT=3] [N=2]	[T=1] [B=1] [R=0] [CB=4] [CT=3] [N=3]	[T=1] [B=1] [R=0] [CB=4] [CT=4] [N=0]	[T=1] [B=1] [R=0] [CB=4] [CT=4] [N=1]
R57	ct_coef															
	[T=1] [B=1] [R=0] [CB=4] [CT=4] [N=2]	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5] [CT=1]	[T=1] [B=1] [R=0] [CB=5] [CT=1]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5]	[T=1] [B=1] [R=0] [CB=5] [CT=2] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=4] [N=3]	[CT=5] [N=0]	[CT=5] [N=1]	[CT=5] [N=2]	[CT=5] [N=3]	[CT=0] [N=0]	[CT=0] [N=1]	[CT=0] [N=2]	[CT=0] [N=3]	[N=0]	[N=1]	[CT=1] [N=2]	[CT=1] [N=3]	[CT=2] [N=0]	
R58	ct_coef															
	[T=1] [B=1] [R=0] [CB=5] [CT=2] [N=2]	[T=1] [B=1] [R=0] [CB=5] [CT=2] [N=3]	[T=1] [B=1] [R=0] [CB=5] [CT=3] [N=0]	[T=1] [B=1] [R=0] [CB=5] [CT=3] [N=1]	[T=1] [B=1] [R=0] [CB=5] [CT=3] [N=2]	[T=1] [B=1] [R=0] [CB=5] [CT=3] [N=3]	[T=1] [B=1] [R=0] [CB=5] [CT=4] [N=0]	[T=1] [B=1] [R=0] [CB=5] [CT=4] [N=1]	[T=1] [B=1] [R=0] [CB=5] [CT=4] [N=2]	[T=1] [B=1] [R=0] [CB=5] [CT=4] [N=3]	[T=1] [B=1] [R=0] [CB=5] [CT=5] [N=0]	[T=1] [B=1] [R=0] [CB=5] [CT=5] [N=1]	[T=1] [B=1] [R=0] [CB=5] [CT=5] [N=2]	[T=1] [B=1] [R=0] [CB=5] [CT=5] [N=3]	[T=1] [B=1] [R=1] [CB=0] [CT=0] [N=0]	[T=1] [B=1] [R=1] [CB=0] [CT=0] [N=1]
R59	ct_coef															
	[T=1] [B=1] [R=1] [CB=0] [CT=0] [N=2]	[T=1] [B=1] [R=1] [CB=0] [CT=0] [N=3]	[T=1] [B=1] [R=1] [CB=0] [CT=1] [N=0]	[T=1] [B=1] [R=1] [CB=0] [CT=1] [N=1]	[T=1] [B=1] [R=1] [CB=0] [CT=1] [N=2]	[T=1] [B=1] [R=1] [CB=0] [CT=1] [N=3]	[T=1] [B=1] [R=1] [CB=0] [CT=2] [N=0]	[T=1] [B=1] [R=1] [CB=0] [CT=2] [N=1]	[T=1] [B=1] [R=1] [CB=0] [CT=2] [N=2]	[T=1] [B=1] [R=1] [CB=0] [CT=2] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=0] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=0] [N=1]	[T=1] [B=1] [R=1] [CB=1] [CT=0] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=0] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=1] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=1] [N=1]
R60	ct_coef															
	[T=1] [B=1] [R=1] [CB=1] [CT=1] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=1] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=2] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=2] [N=1]	[T=1] [B=1] [R=1] [CB=1] [CT=2] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=2] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=3] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=3] [N=1]	[T=1] [B=1] [R=1] [CB=1] [CT=3] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=3] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=4] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=4] [N=1]	[T=1] [B=1] [R=1] [CB=1] [CT=4] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=4] [N=3]	[T=1] [B=1] [R=1] [CB=1] [CT=5] [N=0]	[T=1] [B=1] [R=1] [CB=1] [CT=5] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R61	ct_coef															
	[T=1] [B=1] [R=1] [CB=1] [CT=5] [N=2]	[T=1] [B=1] [R=1] [CB=1] [CT=5] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=0] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=0] [N=1]	[T=1] [B=1] [R=1] [CB=2] [CT=0] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=0] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=1] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=1] [N=1]	[T=1] [B=1] [R=1] [CB=2] [CT=1] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=1] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=2] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=2] [N=1]	[T=1] [B=1] [R=1] [CB=2] [CT=2] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=2] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=3] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=3] [N=1]
R62	ct_coef															
	[T=1] [B=1] [R=1] [CB=2] [CT=3] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=3] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=4] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=4] [N=1]	[T=1] [B=1] [R=1] [CB=2] [CT=4] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=4] [N=3]	[T=1] [B=1] [R=1] [CB=2] [CT=5] [N=0]	[T=1] [B=1] [R=1] [CB=2] [CT=5] [N=1]	[T=1] [B=1] [R=1] [CB=2] [CT=5] [N=2]	[T=1] [B=1] [R=1] [CB=2] [CT=5] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=0] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=0] [N=1]	[T=1] [B=1] [R=1] [CB=3] [CT=0] [N=2]	[T=1] [B=1] [R=1] [CB=3] [CT=0] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=1] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=1] [N=1]
R63	ct_coef															
	[T=1] [B=1] [R=1] [CB=3] [CT=1] [N=2]	[T=1] [B=1] [R=1] [CB=3] [CT=1] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=2] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=2] [N=1]	[T=1] [B=1] [R=1] [CB=3] [CT=2] [N=2]	[T=1] [B=1] [R=1] [CB=3] [CT=2] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=3] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=3] [N=1]	[T=1] [B=1] [R=1] [CB=3] [CT=3] [N=2]	[T=1] [B=1] [R=1] [CB=3] [CT=3] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=4] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=4] [N=1]	[T=1] [B=1] [R=1] [CB=3] [CT=4] [N=2]	[T=1] [B=1] [R=1] [CB=3] [CT=4] [N=3]	[T=1] [B=1] [R=1] [CB=3] [CT=5] [N=0]	[T=1] [B=1] [R=1] [CB=3] [CT=5] [N=1]
R64	ct_coef															
	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]	[T=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=1] [R=1] [CB=3] [CT=5] [N=2]	[B=1] [R=1] [CB=3] [CT=5] [N=3]	[B=1] [R=1] [CB=4] [CT=0] [N=0]	[B=1] [R=1] [CB=4] [CT=0] [N=1]	[B=1] [R=1] [CB=4] [CT=0] [N=2]	[B=1] [R=1] [CB=4] [CT=0] [N=3]	[B=1] [R=1] [CB=4] [CT=1] [N=0]	[B=1] [R=1] [CB=4] [CT=1] [N=1]	[B=1] [R=1] [CB=4] [CT=1] [N=2]	[B=1] [R=1] [CB=4] [CT=1] [N=3]	[B=1] [R=1] [CB=4] [CT=2] [N=0]	[B=1] [R=1] [CB=4] [CT=2] [N=1]	[B=1] [R=1] [CB=4] [CT=2] [N=2]	[B=1] [R=1] [CB=4] [CT=2] [N=3]	[B=1] [R=1] [CB=4] [CT=3] [N=0]	[B=1] [R=1] [CB=4] [CT=3] [N=1]
R65	ct_coef															
	[T=1] [B=1] [R=1] [CB=4] [CT=3] [N=2]	[T=1] [B=1] [R=1] [CB=4] [CT=3] [N=3]	[T=1] [B=1] [R=1] [CB=4] [CT=4] [N=0]	[T=1] [B=1] [R=1] [CB=4] [CT=4] [N=1]	[T=1] [B=1] [R=1] [CB=4] [CT=4] [N=2]	[T=1] [B=1] [R=1] [CB=4] [CT=4] [N=3]	[T=1] [B=1] [R=1] [CB=4] [CT=5] [N=0]	[T=1] [B=1] [R=1] [CB=4] [CT=5] [N=1]	[T=1] [B=1] [R=1] [CB=4] [CT=5] [N=2]	[T=1] [B=1] [R=1] [CB=4] [CT=5] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=0] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=0] [N=1]	[T=1] [B=1] [R=1] [CB=5] [CT=0] [N=2]	[T=1] [B=1] [R=1] [CB=5] [CT=0] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=1] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=1] [N=1]
R66	ct_coef															
	[T=1] [B=1] [R=1] [CB=5] [CT=1] [N=2]	[T=1] [B=1] [R=1] [CB=5] [CT=1] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=2] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=2] [N=1]	[T=1] [B=1] [R=1] [CB=5] [CT=2] [N=2]	[T=1] [B=1] [R=1] [CB=5] [CT=2] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=3] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=3] [N=1]	[T=1] [B=1] [R=1] [CB=5] [CT=3] [N=2]	[T=1] [B=1] [R=1] [CB=5] [CT=3] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=4] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=4] [N=1]	[T=1] [B=1] [R=1] [CB=5] [CT=4] [N=2]	[T=1] [B=1] [R=1] [CB=5] [CT=4] [N=3]	[T=1] [B=1] [R=1] [CB=5] [CT=5] [N=0]	[T=1] [B=1] [R=1] [CB=5] [CT=5] [N=1]
R67	ct_coef															
	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]	[T=2] [B=0] [R=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=5] [CT=5] [N=2]	[CB=5] [CT=5] [N=3]	[CB=0] [CT=0] [N=0]	[CB=0] [CT=0] [N=1]	[CB=0] [CT=0] [N=2]	[CB=0] [CT=0] [N=3]	[CB=0] [CT=1] [N=0]	[CB=0] [CT=1] [N=1]	[CB=0] [CT=1] [N=2]	[CB=0] [CT=1] [N=3]	[CB=0] [CT=2] [N=0]	[CB=0] [CT=2] [N=1]	[CB=0] [CT=2] [N=2]	[CB=0] [CT=2] [N=3]	[CB=1] [CT=0] [N=0]	[CB=1] [CT=0] [N=1]
R68	ct_coef															
	[T=2] [B=0] [R=0] [CB=1] [CT=0] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=0] [N=3]	[T=2] [B=0] [R=0] [CB=1] [CT=1] [N=0]	[T=2] [B=0] [R=0] [CB=1] [CT=1] [N=1]	[T=2] [B=0] [R=0] [CB=1] [CT=1] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=1] [N=3]	[T=2] [B=0] [R=0] [CB=1] [CT=2] [N=0]	[T=2] [B=0] [R=0] [CB=1] [CT=2] [N=1]	[T=2] [B=0] [R=0] [CB=1] [CT=2] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=2] [N=3]	[T=2] [B=0] [R=0] [CB=1] [CT=3] [N=0]	[T=2] [B=0] [R=0] [CB=1] [CT=3] [N=1]	[T=2] [B=0] [R=0] [CB=1] [CT=3] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=3] [N=3]	[T=2] [B=0] [R=0] [CB=1] [CT=4] [N=0]	[T=2] [B=0] [R=0] [CB=1] [CT=4] [N=1]
R69	ct_coef															
	[T=2] [B=0] [R=0] [CB=1] [CT=4] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=4] [N=3]	[T=2] [B=0] [R=0] [CB=1] [CT=5] [N=0]	[T=2] [B=0] [R=0] [CB=1] [CT=5] [N=1]	[T=2] [B=0] [R=0] [CB=1] [CT=5] [N=2]	[T=2] [B=0] [R=0] [CB=1] [CT=5] [N=3]	[T=2] [B=0] [R=0] [CB=2] [CT=0] [N=0]	[T=2] [B=0] [R=0] [CB=2] [CT=0] [N=1]	[T=2] [B=0] [R=0] [CB=2] [CT=0] [N=2]	[T=2] [B=0] [R=0] [CB=2] [CT=0] [N=3]	[T=2] [B=0] [R=0] [CB=2] [CT=1] [N=0]	[T=2] [B=0] [R=0] [CB=2] [CT=1] [N=1]	[T=2] [B=0] [R=0] [CB=2] [CT=1] [N=2]	[T=2] [B=0] [R=0] [CB=2] [CT=1] [N=3]	[T=2] [B=0] [R=0] [CB=2] [CT=2] [N=0]	[T=2] [B=0] [R=0] [CB=2] [CT=2] [N=1]
R70	ct_coef															
	[T=2] [B=0] [R=0] [CB=2] [CT=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=2]	[T=2] [B=0] [R=0] [CB=3]	[T=2] [B=0] [R=0] [CB=3] [CT=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[N=2]	[CT=2] [N=3]	[CT=3] [N=0]	[CT=3] [N=1]	[CT=3] [N=2]	[CT=3] [N=3]	[CT=4] [N=0]	[CT=4] [N=1]	[CT=4] [N=2]	[CT=4] [N=3]	[CT=5] [N=0]	[CT=5] [N=1]	[CT=5] [N=2]	[CT=5] [N=3]	[CT=0] [N=0]	[N=1]
R71	ct_coef															
	[T=2] [B=0] [R=0] [CB=3] [CT=0] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=0] [N=3]	[T=2] [B=0] [R=0] [CB=3] [CT=1] [N=0]	[T=2] [B=0] [R=0] [CB=3] [CT=1] [N=1]	[T=2] [B=0] [R=0] [CB=3] [CT=1] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=1] [N=3]	[T=2] [B=0] [R=0] [CB=3] [CT=2] [N=0]	[T=2] [B=0] [R=0] [CB=3] [CT=2] [N=1]	[T=2] [B=0] [R=0] [CB=3] [CT=2] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=2] [N=3]	[T=2] [B=0] [R=0] [CB=3] [CT=3] [N=0]	[T=2] [B=0] [R=0] [CB=3] [CT=3] [N=1]	[T=2] [B=0] [R=0] [CB=3] [CT=3] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=3] [N=3]	[T=2] [B=0] [R=0] [CB=3] [CT=4] [N=0]	[T=2] [B=0] [R=0] [CB=3] [CT=4] [N=1]
R72	ct_coef															
	[T=2] [B=0] [R=0] [CB=3] [CT=4] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=4] [N=3]	[T=2] [B=0] [R=0] [CB=3] [CT=5] [N=0]	[T=2] [B=0] [R=0] [CB=3] [CT=5] [N=1]	[T=2] [B=0] [R=0] [CB=3] [CT=5] [N=2]	[T=2] [B=0] [R=0] [CB=3] [CT=5] [N=3]	[T=2] [B=0] [R=0] [CB=4] [CT=0] [N=0]	[T=2] [B=0] [R=0] [CB=4] [CT=0] [N=1]	[T=2] [B=0] [R=0] [CB=4] [CT=0] [N=2]	[T=2] [B=0] [R=0] [CB=4] [CT=0] [N=3]	[T=2] [B=0] [R=0] [CB=4] [CT=1] [N=0]	[T=2] [B=0] [R=0] [CB=4] [CT=1] [N=1]	[T=2] [B=0] [R=0] [CB=4] [CT=1] [N=2]	[T=2] [B=0] [R=0] [CB=4] [CT=1] [N=3]	[T=2] [B=0] [R=0] [CB=4] [CT=2] [N=0]	[T=2] [B=0] [R=0] [CB=4] [CT=2] [N=1]
R73	ct_coef															
	[T=2] [B=0] [R=0] [CB=4] [CT=2] [N=2]	[T=2] [B=0] [R=0] [CB=4] [CT=2]	[T=2] [B=0] [R=0] [CB=4] [CT=3]	[T=2] [B=0] [R=0] [CB=4] [CT=3]	[T=2] [B=0] [R=0] [CB=4] [CT=3]	[T=2] [B=0] [R=0] [CB=4] [CT=3]	[T=2] [B=0] [R=0] [CB=4] [CT=4]	[T=2] [B=0] [R=0] [CB=4] [CT=4]	[T=2] [B=0] [R=0] [CB=4] [CT=4]	[T=2] [B=0] [R=0] [CB=4] [CT=4]	[T=2] [B=0] [R=0] [CB=4] [CT=5] [N=0]	[T=2] [B=0] [R=0] [CB=4] [CT=5] [N=1]	[T=2] [B=0] [R=0] [CB=4] [CT=5]	[T=2] [B=0] [R=0] [CB=4] [CT=5]	[T=2] [B=0] [R=0] [CB=5] [CT=0]	[T=2] [B=0] [R=0] [CB=5] [CT=0] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[N=3]	[N=0]	[N=1]	[N=2]	[N=3]	[N=0]	[N=1]	[N=2]	[N=3]			[N=2]	[N=3]	[N=0]	
R74	ct_coef															
	[T=2] [B=0] [R=0] [CB=5] [CT=0] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=0] [N=3]	[T=2] [B=0] [R=0] [CB=5] [CT=1] [N=0]	[T=2] [B=0] [R=0] [CB=5] [CT=1] [N=1]	[T=2] [B=0] [R=0] [CB=5] [CT=1] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=1] [N=3]	[T=2] [B=0] [R=0] [CB=5] [CT=2] [N=0]	[T=2] [B=0] [R=0] [CB=5] [CT=2] [N=1]	[T=2] [B=0] [R=0] [CB=5] [CT=2] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=2] [N=3]	[T=2] [B=0] [R=0] [CB=5] [CT=3] [N=0]	[T=2] [B=0] [R=0] [CB=5] [CT=3] [N=1]	[T=2] [B=0] [R=0] [CB=5] [CT=3] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=3] [N=3]	[T=2] [B=0] [R=0] [CB=5] [CT=4] [N=0]	[T=2] [B=0] [R=0] [CB=5] [CT=4] [N=1]
R75	ct_coef															
	[T=2] [B=0] [R=0] [CB=5] [CT=4] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=4] [N=3]	[T=2] [B=0] [R=0] [CB=5] [CT=5] [N=0]	[T=2] [B=0] [R=0] [CB=5] [CT=5] [N=1]	[T=2] [B=0] [R=0] [CB=5] [CT=5] [N=2]	[T=2] [B=0] [R=0] [CB=5] [CT=5] [N=3]	[T=2] [B=0] [R=1] [CB=0] [CT=0] [N=0]	[T=2] [B=0] [R=1] [CB=0] [CT=0] [N=1]	[T=2] [B=0] [R=1] [CB=0] [CT=0] [N=2]	[T=2] [B=0] [R=1] [CB=0] [CT=0] [N=3]	[T=2] [B=0] [R=1] [CB=0] [CT=1] [N=0]	[T=2] [B=0] [R=1] [CB=0] [CT=1] [N=1]	[T=2] [B=0] [R=1] [CB=0] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=0] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=0] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=0] [CT=2] [N=1]
R76	ct_coef															
	[T=2] [B=0] [R=1] [CB=0] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=0] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=0] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=0] [N=1]	[T=2] [B=0] [R=1] [CB=1] [CT=0] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=0] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=1] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=1] [N=1]	[T=2] [B=0] [R=1] [CB=1] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=2] [N=1]	[T=2] [B=0] [R=1] [CB=1] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=3] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=3] [N=1]
R77	ct_coef															

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[T=2] [B=0] [R=1] [CB=1] [CT=3] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=3] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=4] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=4] [N=1]	[T=2] [B=0] [R=1] [CB=1] [CT=4] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=4] [N=3]	[T=2] [B=0] [R=1] [CB=1] [CT=5] [N=0]	[T=2] [B=0] [R=1] [CB=1] [CT=5] [N=1]	[T=2] [B=0] [R=1] [CB=1] [CT=5] [N=2]	[T=2] [B=0] [R=1] [CB=1] [CT=5] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=0] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=0] [N=1]	[T=2] [B=0] [R=1] [CB=2] [CT=0] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=0] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=1] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=1] [N=1]
R78	ct_coef															
	[T=2] [B=0] [R=1] [CB=2] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=2] [N=1]	[T=2] [B=0] [R=1] [CB=2] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=3] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=3] [N=1]	[T=2] [B=0] [R=1] [CB=2] [CT=3] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=3] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=4] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=4] [N=1]	[T=2] [B=0] [R=1] [CB=2] [CT=4] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=4] [N=3]	[T=2] [B=0] [R=1] [CB=2] [CT=5] [N=0]	[T=2] [B=0] [R=1] [CB=2] [CT=5] [N=1]
R79	ct_coef															
	[T=2] [B=0] [R=1] [CB=2] [CT=5] [N=2]	[T=2] [B=0] [R=1] [CB=2] [CT=5] [N=3]	[T=2] [B=0] [R=1] [CB=3] [CT=0] [N=0]	[T=2] [B=0] [R=1] [CB=3] [CT=0] [N=1]	[T=2] [B=0] [R=1] [CB=3] [CT=0] [N=2]	[T=2] [B=0] [R=1] [CB=3] [CT=0] [N=3]	[T=2] [B=0] [R=1] [CB=3] [CT=1] [N=0]	[T=2] [B=0] [R=1] [CB=3] [CT=1] [N=1]	[T=2] [B=0] [R=1] [CB=3] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=3] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=3] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=3] [CT=2] [N=1]	[T=2] [B=0] [R=1] [CB=3] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=3] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=3] [CT=3] [N=0]	[T=2] [B=0] [R=1] [CB=3] [CT=3] [N=1]
R80	ct_coef															
	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]	[T=2] [B=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[R=1] [CB=3] [CT=3] [N=2]	[R=1] [CB=3] [CT=3] [N=3]	[R=1] [CB=3] [CT=4] [N=0]	[R=1] [CB=3] [CT=4] [N=1]	[R=1] [CB=3] [CT=4] [N=2]	[R=1] [CB=3] [CT=4] [N=3]	[R=1] [CB=3] [CT=5] [N=0]	[R=1] [CB=3] [CT=5] [N=1]	[R=1] [CB=3] [CT=5] [N=2]	[R=1] [CB=3] [CT=5] [N=3]	[R=1] [CB=4] [CT=0] [N=0]	[R=1] [CB=4] [CT=0] [N=1]	[R=1] [CB=4] [CT=0] [N=2]	[R=1] [CB=4] [CT=0] [N=3]	[R=1] [CB=4] [CT=1] [N=0]	[R=1] [CB=4] [CT=1] [N=1]
R81	ct_coef															
	[T=2] [B=0] [R=1] [CB=4] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=4] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=4] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=4] [CT=2] [N=1]	[T=2] [B=0] [R=1] [CB=4] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=4] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=4] [CT=3] [N=0]	[T=2] [B=0] [R=1] [CB=4] [CT=3] [N=1]	[T=2] [B=0] [R=1] [CB=4] [CT=3] [N=2]	[T=2] [B=0] [R=1] [CB=4] [CT=3] [N=3]	[T=2] [B=0] [R=1] [CB=4] [CT=4] [N=0]	[T=2] [B=0] [R=1] [CB=4] [CT=4] [N=1]	[T=2] [B=0] [R=1] [CB=4] [CT=4] [N=2]	[T=2] [B=0] [R=1] [CB=4] [CT=4] [N=3]	[T=2] [B=0] [R=1] [CB=4] [CT=5] [N=0]	[T=2] [B=0] [R=1] [CB=4] [CT=5] [N=1]
R82	ct_coef															
	[T=2] [B=0] [R=1] [CB=4] [CT=5] [N=2]	[T=2] [B=0] [R=1] [CB=4] [CT=5] [N=3]	[T=2] [B=0] [R=1] [CB=5] [CT=0] [N=0]	[T=2] [B=0] [R=1] [CB=5] [CT=0] [N=1]	[T=2] [B=0] [R=1] [CB=5] [CT=0] [N=2]	[T=2] [B=0] [R=1] [CB=5] [CT=0] [N=3]	[T=2] [B=0] [R=1] [CB=5] [CT=1] [N=0]	[T=2] [B=0] [R=1] [CB=5] [CT=1] [N=1]	[T=2] [B=0] [R=1] [CB=5] [CT=1] [N=2]	[T=2] [B=0] [R=1] [CB=5] [CT=1] [N=3]	[T=2] [B=0] [R=1] [CB=5] [CT=2] [N=0]	[T=2] [B=0] [R=1] [CB=5] [CT=2] [N=1]	[T=2] [B=0] [R=1] [CB=5] [CT=2] [N=2]	[T=2] [B=0] [R=1] [CB=5] [CT=2] [N=3]	[T=2] [B=0] [R=1] [CB=5] [CT=3] [N=0]	[T=2] [B=0] [R=1] [CB=5] [CT=3] [N=1]
R83	ct_coef															
	[T=2] [B=0] [R=1] [CB=5]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=0] [R=1]	[T=2] [B=1] [R=0]	[T=2] [B=1] [R=0]	[T=2] [B=1] [R=0]	[T=2] [B=1] [R=0]	[T=2] [B=1] [R=0]	[T=2] [B=1] [R=0] [CB=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CT=3] [N=2]	[CB=5] [CT=3] [N=3]	[CB=5] [CT=4] [N=0]	[CB=5] [CT=4] [N=1]	[CB=5] [CT=4] [N=2]	[CB=5] [CT=4] [N=3]	[CB=5] [CT=5] [N=0]	[CB=5] [CT=5] [N=1]	[CB=5] [CT=5] [N=2]	[CB=5] [CT=5] [N=3]	[CB=0] [CT=0] [N=0]	[CB=0] [CT=0] [N=1]	[CB=0] [CT=0] [N=2]	[CB=0] [CT=0] [N=3]	[CB=0] [CT=1] [N=0]	[CT=1] [N=1]
R84	ct_coef															
	[T=2] [B=1] [R=0] [CB=0] [CT=1] [N=2]	[T=2] [B=1] [R=0] [CB=0] [CT=1] [N=3]	[T=2] [B=1] [R=0] [CB=0] [CT=2] [N=0]	[T=2] [B=1] [R=0] [CB=0] [CT=2] [N=1]	[T=2] [B=1] [R=0] [CB=0] [CT=2] [N=2]	[T=2] [B=1] [R=0] [CB=0] [CT=2] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=0] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=0] [N=1]	[T=2] [B=1] [R=0] [CB=1] [CT=0] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=0] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=1] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=1] [N=1]	[T=2] [B=1] [R=0] [CB=1] [CT=1] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=1] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=2] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=2] [N=1]
R85	ct_coef															
	[T=2] [B=1] [R=0] [CB=1] [CT=2] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=2] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=3] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=3] [N=1]	[T=2] [B=1] [R=0] [CB=1] [CT=3] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=3] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=4] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=4] [N=1]	[T=2] [B=1] [R=0] [CB=1] [CT=4] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=4] [N=3]	[T=2] [B=1] [R=0] [CB=1] [CT=5] [N=0]	[T=2] [B=1] [R=0] [CB=1] [CT=5] [N=1]	[T=2] [B=1] [R=0] [CB=1] [CT=5] [N=2]	[T=2] [B=1] [R=0] [CB=1] [CT=5] [N=3]	[T=2] [B=1] [R=0] [CB=2] [CT=0] [N=0]	[T=2] [B=1] [R=0] [CB=2] [CT=0] [N=1]
R86	ct_coef															
	[T=2] [B=1] [R=0] [CB=2] [CT=0] [N=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2] [CT=3]	[T=2] [B=1] [R=0] [CB=2] [CT=3]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2] [CT=4] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=0] [N=3]	[CT=1] [N=0]	[CT=1] [N=1]	[CT=1] [N=2]	[CT=1] [N=3]	[CT=2] [N=0]	[CT=2] [N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[N=0]	[N=1]	[CT=3] [N=2]	[CT=3] [N=3]	[CT=4] [N=0]	
R87	ct_coef															
	[T=2] [B=1] [R=0] [CB=2] [CT=4] [N=2]	[T=2] [B=1] [R=0] [CB=2] [CT=4] [N=3]	[T=2] [B=1] [R=0] [CB=2] [CT=5] [N=0]	[T=2] [B=1] [R=0] [CB=2] [CT=5] [N=1]	[T=2] [B=1] [R=0] [CB=2] [CT=5] [N=2]	[T=2] [B=1] [R=0] [CB=2] [CT=5] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=0] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=0] [N=1]	[T=2] [B=1] [R=0] [CB=3] [CT=0] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=0] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=1] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=1] [N=1]	[T=2] [B=1] [R=0] [CB=3] [CT=1] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=1] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=2] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=2] [N=1]
R88	ct_coef															
	[T=2] [B=1] [R=0] [CB=3] [CT=2] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=2] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=3] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=3] [N=1]	[T=2] [B=1] [R=0] [CB=3] [CT=3] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=3] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=4] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=4] [N=1]	[T=2] [B=1] [R=0] [CB=3] [CT=4] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=4] [N=3]	[T=2] [B=1] [R=0] [CB=3] [CT=5] [N=0]	[T=2] [B=1] [R=0] [CB=3] [CT=5] [N=1]	[T=2] [B=1] [R=0] [CB=3] [CT=5] [N=2]	[T=2] [B=1] [R=0] [CB=3] [CT=5] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=0] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=0] [N=1]
R89	ct_coef															
	[T=2] [B=1] [R=0] [CB=4] [CT=0] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=0] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=1] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=1] [N=1]	[T=2] [B=1] [R=0] [CB=4] [CT=1] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=1] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=2] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=2] [N=1]	[T=2] [B=1] [R=0] [CB=4] [CT=2] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=2] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=3] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=3] [N=1]	[T=2] [B=1] [R=0] [CB=4] [CT=3] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=3] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=4] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=4] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R90	ct_coef															
	[T=2] [B=1] [R=0] [CB=4] [CT=4] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=4] [N=3]	[T=2] [B=1] [R=0] [CB=4] [CT=5] [N=0]	[T=2] [B=1] [R=0] [CB=4] [CT=5] [N=1]	[T=2] [B=1] [R=0] [CB=4] [CT=5] [N=2]	[T=2] [B=1] [R=0] [CB=4] [CT=5] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=0] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=0] [N=1]	[T=2] [B=1] [R=0] [CB=5] [CT=0] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=0] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=1] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=1] [N=1]	[T=2] [B=1] [R=0] [CB=5] [CT=1] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=1] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=2] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=2] [N=1]
R91	ct_coef															
	[T=2] [B=1] [R=0] [CB=5] [CT=2] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=2] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=3] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=3] [N=1]	[T=2] [B=1] [R=0] [CB=5] [CT=3] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=3] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=4] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=4] [N=1]	[T=2] [B=1] [R=0] [CB=5] [CT=4] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=4] [N=3]	[T=2] [B=1] [R=0] [CB=5] [CT=5] [N=0]	[T=2] [B=1] [R=0] [CB=5] [CT=5] [N=1]	[T=2] [B=1] [R=0] [CB=5] [CT=5] [N=2]	[T=2] [B=1] [R=0] [CB=5] [CT=5] [N=3]	[T=2] [B=1] [R=1] [CB=0] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=0] [CT=0] [N=1]
R92	ct_coef															
	[T=2] [B=1] [R=1] [CB=0] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=0] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=0] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=0] [CT=1] [N=1]	[T=2] [B=1] [R=1] [CB=0] [CT=1] [N=2]	[T=2] [B=1] [R=1] [CB=0] [CT=1] [N=3]	[T=2] [B=1] [R=1] [CB=0] [CT=2] [N=0]	[T=2] [B=1] [R=1] [CB=0] [CT=2] [N=1]	[T=2] [B=1] [R=1] [CB=0] [CT=2] [N=2]	[T=2] [B=1] [R=1] [CB=0] [CT=2] [N=3]	[T=2] [B=1] [R=1] [CB=1] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=1] [CT=0] [N=1]	[T=2] [B=1] [R=1] [CB=1] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=1] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=1] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=1] [CT=1] [N=1]
R93	ct_coef															
	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=1] [R=1] [CB=1] [CT=1] [N=2]	[B=1] [R=1] [CB=1] [CT=1] [N=3]	[B=1] [R=1] [CB=1] [CT=2] [N=0]	[B=1] [R=1] [CB=1] [CT=2] [N=1]	[B=1] [R=1] [CB=1] [CT=2] [N=2]	[B=1] [R=1] [CB=1] [CT=2] [N=3]	[B=1] [R=1] [CB=1] [CT=3] [N=0]	[B=1] [R=1] [CB=1] [CT=3] [N=1]	[B=1] [R=1] [CB=1] [CT=3] [N=2]	[B=1] [R=1] [CB=1] [CT=3] [N=3]	[B=1] [R=1] [CB=1] [CT=4] [N=0]	[B=1] [R=1] [CB=1] [CT=4] [N=1]	[B=1] [R=1] [CB=1] [CT=4] [N=2]	[B=1] [R=1] [CB=1] [CT=4] [N=3]	[B=1] [R=1] [CB=1] [CT=5] [N=0]	[B=1] [R=1] [CB=1] [CT=5] [N=1]
R94	ct_coef															
	[T=2] [B=1] [R=1] [CB=1] [CT=5] [N=2]	[T=2] [B=1] [R=1] [CB=1] [CT=5] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=0] [N=1]	[T=2] [B=1] [R=1] [CB=2] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=1] [N=1]	[T=2] [B=1] [R=1] [CB=2] [CT=1] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=1] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=2] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=2] [N=1]	[T=2] [B=1] [R=1] [CB=2] [CT=2] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=2] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=3] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=3] [N=1]
R95	ct_coef															
	[T=2] [B=1] [R=1] [CB=2] [CT=3] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=3] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=4] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=4] [N=1]	[T=2] [B=1] [R=1] [CB=2] [CT=4] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=4] [N=3]	[T=2] [B=1] [R=1] [CB=2] [CT=5] [N=0]	[T=2] [B=1] [R=1] [CB=2] [CT=5] [N=1]	[T=2] [B=1] [R=1] [CB=2] [CT=5] [N=2]	[T=2] [B=1] [R=1] [CB=2] [CT=5] [N=3]	[T=2] [B=1] [R=1] [CB=3] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=3] [CT=0] [N=1]	[T=2] [B=1] [R=1] [CB=3] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=3] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=3] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=3] [CT=1] [N=1]
R96	ct_coef															
	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]	[T=2] [B=1] [R=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=3] [CT=1] [N=2]	[CB=3] [CT=1] [N=3]	[CB=3] [CT=2] [N=0]	[CB=3] [CT=2] [N=1]	[CB=3] [CT=2] [N=2]	[CB=3] [CT=2] [N=3]	[CB=3] [CT=3] [N=0]	[CB=3] [CT=3] [N=1]	[CB=3] [CT=3] [N=2]	[CB=3] [CT=3] [N=3]	[CB=3] [CT=4] [N=0]	[CB=3] [CT=4] [N=1]	[CB=3] [CT=4] [N=2]	[CB=3] [CT=4] [N=3]	[CB=3] [CT=5] [N=0]	[CB=3] [CT=5] [N=1]
R97	ct_coef															
	[T=2] [B=1] [R=1] [CB=3] [CT=5] [N=2]	[T=2] [B=1] [R=1] [CB=3] [CT=5] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=0] [N=1]	[T=2] [B=1] [R=1] [CB=4] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=1] [N=1]	[T=2] [B=1] [R=1] [CB=4] [CT=1] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=1] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=2] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=2] [N=1]	[T=2] [B=1] [R=1] [CB=4] [CT=2] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=2] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=3] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=3] [N=1]
R98	ct_coef															
	[T=2] [B=1] [R=1] [CB=4] [CT=3] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=3] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=4] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=4] [N=1]	[T=2] [B=1] [R=1] [CB=4] [CT=4] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=4] [N=3]	[T=2] [B=1] [R=1] [CB=4] [CT=5] [N=0]	[T=2] [B=1] [R=1] [CB=4] [CT=5] [N=1]	[T=2] [B=1] [R=1] [CB=4] [CT=5] [N=2]	[T=2] [B=1] [R=1] [CB=4] [CT=5] [N=3]	[T=2] [B=1] [R=1] [CB=5] [CT=0] [N=0]	[T=2] [B=1] [R=1] [CB=5] [CT=0] [N=1]	[T=2] [B=1] [R=1] [CB=5] [CT=0] [N=2]	[T=2] [B=1] [R=1] [CB=5] [CT=0] [N=3]	[T=2] [B=1] [R=1] [CB=5] [CT=1] [N=0]	[T=2] [B=1] [R=1] [CB=5] [CT=1] [N=1]
R99	ct_coef															
	[T=2] [B=1] [R=1] [CB=5] [CT=1]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5]	[T=2] [B=1] [R=1] [CB=5] [CT=5]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[N=2]	[CT=1] [N=3]	[CT=2] [N=0]	[CT=2] [N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[CT=3] [N=0]	[CT=3] [N=1]	[CT=3] [N=2]	[CT=3] [N=3]	[CT=4] [N=0]	[CT=4] [N=1]	[CT=4] [N=2]	[CT=4] [N=3]	[CT=5] [N=0]	[N=1]
R100	ct_coef															
	[T=2] [B=1] [R=1] [CB=5] [CT=5] [N=2]	[T=2] [B=1] [R=1] [CB=5] [CT=5] [N=3]	[T=3] [B=0] [R=0] [CB=0] [CT=0] [N=0]	[T=3] [B=0] [R=0] [CB=0] [CT=0] [N=1]	[T=3] [B=0] [R=0] [CB=0] [CT=0] [N=2]	[T=3] [B=0] [R=0] [CB=0] [CT=0] [N=3]	[T=3] [B=0] [R=0] [CB=0] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=0] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=0] [CT=1] [N=2]	[T=3] [B=0] [R=0] [CB=0] [CT=1] [N=3]	[T=3] [B=0] [R=0] [CB=0] [CT=2] [N=0]	[T=3] [B=0] [R=0] [CB=0] [CT=2] [N=1]	[T=3] [B=0] [R=0] [CB=0] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=0] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=1] [CT=0] [N=0]	[T=3] [B=0] [R=0] [CB=1] [CT=0] [N=1]
R101	ct_coef															
	[T=3] [B=0] [R=0] [CB=1] [CT=0] [N=2]	[T=3] [B=0] [R=0] [CB=1] [CT=0] [N=3]	[T=3] [B=0] [R=0] [CB=1] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=1] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=1] [CT=1] [N=2]	[T=3] [B=0] [R=0] [CB=1] [CT=1] [N=3]	[T=3] [B=0] [R=0] [CB=1] [CT=2] [N=0]	[T=3] [B=0] [R=0] [CB=1] [CT=2] [N=1]	[T=3] [B=0] [R=0] [CB=1] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=1] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=1] [CT=3] [N=0]	[T=3] [B=0] [R=0] [CB=1] [CT=3] [N=1]	[T=3] [B=0] [R=0] [CB=1] [CT=3] [N=2]	[T=3] [B=0] [R=0] [CB=1] [CT=3] [N=3]	[T=3] [B=0] [R=0] [CB=1] [CT=4] [N=0]	[T=3] [B=0] [R=0] [CB=1] [CT=4] [N=1]
R102	ct_coef															
	[T=3] [B=0] [R=0] [CB=1] [CT=4] [N=2]	[T=3] [B=0] [R=0] [CB=1] [CT=4]	[T=3] [B=0] [R=0] [CB=1] [CT=5]	[T=3] [B=0] [R=0] [CB=1] [CT=5]	[T=3] [B=0] [R=0] [CB=1] [CT=5]	[T=3] [B=0] [R=0] [CB=1] [CT=5]	[T=3] [B=0] [R=0] [CB=2] [CT=0]	[T=3] [B=0] [R=0] [CB=2] [CT=0]	[T=3] [B=0] [R=0] [CB=2] [CT=0]	[T=3] [B=0] [R=0] [CB=2] [CT=0]	[T=3] [B=0] [R=0] [CB=2] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=2] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=2] [CT=1]	[T=3] [B=0] [R=0] [CB=2] [CT=1]	[T=3] [B=0] [R=0] [CB=2] [CT=2]	[T=3] [B=0] [R=0] [CB=2] [CT=2] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[N=3]	[N=0]	[N=1]	[N=2]	[N=3]	[N=0]	[N=1]	[N=2]	[N=3]			[N=2]	[N=3]	[N=0]	
R103	ct_coef															
	[T=3] [B=0] [R=0] [CB=2] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=2] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=2] [CT=3] [N=0]	[T=3] [B=0] [R=0] [CB=2] [CT=3] [N=1]	[T=3] [B=0] [R=0] [CB=2] [CT=3] [N=2]	[T=3] [B=0] [R=0] [CB=2] [CT=3] [N=3]	[T=3] [B=0] [R=0] [CB=2] [CT=4] [N=0]	[T=3] [B=0] [R=0] [CB=2] [CT=4] [N=1]	[T=3] [B=0] [R=0] [CB=2] [CT=4] [N=2]	[T=3] [B=0] [R=0] [CB=2] [CT=4] [N=3]	[T=3] [B=0] [R=0] [CB=2] [CT=5] [N=0]	[T=3] [B=0] [R=0] [CB=2] [CT=5] [N=1]	[T=3] [B=0] [R=0] [CB=2] [CT=5] [N=2]	[T=3] [B=0] [R=0] [CB=2] [CT=5] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=0] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=0] [N=1]
R104	ct_coef															
	[T=3] [B=0] [R=0] [CB=3] [CT=0] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=0] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=3] [CT=1] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=1] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=2] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=2] [N=1]	[T=3] [B=0] [R=0] [CB=3] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=3] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=3] [N=1]	[T=3] [B=0] [R=0] [CB=3] [CT=3] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=3] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=4] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=4] [N=1]
R105	ct_coef															
	[T=3] [B=0] [R=0] [CB=3] [CT=4] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=4] [N=3]	[T=3] [B=0] [R=0] [CB=3] [CT=5] [N=0]	[T=3] [B=0] [R=0] [CB=3] [CT=5] [N=1]	[T=3] [B=0] [R=0] [CB=3] [CT=5] [N=2]	[T=3] [B=0] [R=0] [CB=3] [CT=5] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=0] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=0] [N=1]	[T=3] [B=0] [R=0] [CB=4] [CT=0] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=0] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=4] [CT=1] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=1] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=2] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=2] [N=1]
R106	ct_coef															

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[T=3] [B=0] [R=0] [CB=4] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=3] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=3] [N=1]	[T=3] [B=0] [R=0] [CB=4] [CT=3] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=3] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=4] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=4] [N=1]	[T=3] [B=0] [R=0] [CB=4] [CT=4] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=4] [N=3]	[T=3] [B=0] [R=0] [CB=4] [CT=5] [N=0]	[T=3] [B=0] [R=0] [CB=4] [CT=5] [N=1]	[T=3] [B=0] [R=0] [CB=4] [CT=5] [N=2]	[T=3] [B=0] [R=0] [CB=4] [CT=5] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=0] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=0] [N=1]
R107	ct_coef															
	[T=3] [B=0] [R=0] [CB=5] [CT=0] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=0] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=1] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=1] [N=1]	[T=3] [B=0] [R=0] [CB=5] [CT=1] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=1] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=2] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=2] [N=1]	[T=3] [B=0] [R=0] [CB=5] [CT=2] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=2] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=3] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=3] [N=1]	[T=3] [B=0] [R=0] [CB=5] [CT=3] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=3] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=4] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=4] [N=1]
R108	ct_coef															
	[T=3] [B=0] [R=0] [CB=5] [CT=4] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=4] [N=3]	[T=3] [B=0] [R=0] [CB=5] [CT=5] [N=0]	[T=3] [B=0] [R=0] [CB=5] [CT=5] [N=1]	[T=3] [B=0] [R=0] [CB=5] [CT=5] [N=2]	[T=3] [B=0] [R=0] [CB=5] [CT=5] [N=3]	[T=3] [B=0] [R=1] [CB=0] [CT=0] [N=0]	[T=3] [B=0] [R=1] [CB=0] [CT=0] [N=1]	[T=3] [B=0] [R=1] [CB=0] [CT=0] [N=2]	[T=3] [B=0] [R=1] [CB=0] [CT=0] [N=3]	[T=3] [B=0] [R=1] [CB=0] [CT=1] [N=0]	[T=3] [B=0] [R=1] [CB=0] [CT=1] [N=1]	[T=3] [B=0] [R=1] [CB=0] [CT=1] [N=2]	[T=3] [B=0] [R=1] [CB=0] [CT=1] [N=3]	[T=3] [B=0] [R=1] [CB=0] [CT=2] [N=0]	[T=3] [B=0] [R=1] [CB=0] [CT=2] [N=1]
R109	ct_coef															
	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]	[T=3] [B=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[R=1] [CB=0] [CT=2] [N=2]	[R=1] [CB=0] [CT=2] [N=3]	[R=1] [CB=1] [CT=0] [N=0]	[R=1] [CB=1] [CT=0] [N=1]	[R=1] [CB=1] [CT=0] [N=2]	[R=1] [CB=1] [CT=0] [N=3]	[R=1] [CB=1] [CT=1] [N=0]	[R=1] [CB=1] [CT=1] [N=1]	[R=1] [CB=1] [CT=1] [N=2]	[R=1] [CB=1] [CT=1] [N=3]	[R=1] [CB=1] [CT=2] [N=0]	[R=1] [CB=1] [CT=2] [N=1]	[R=1] [CB=1] [CT=2] [N=2]	[R=1] [CB=1] [CT=2] [N=3]	[R=1] [CB=1] [CT=3] [N=0]	[R=1] [CB=1] [CT=3] [N=1]
R110	ct_coef															
	[T=3] [B=0] [R=1] [CB=1] [CT=3] [N=2]	[T=3] [B=0] [R=1] [CB=1] [CT=3] [N=3]	[T=3] [B=0] [R=1] [CB=1] [CT=4] [N=0]	[T=3] [B=0] [R=1] [CB=1] [CT=4] [N=1]	[T=3] [B=0] [R=1] [CB=1] [CT=4] [N=2]	[T=3] [B=0] [R=1] [CB=1] [CT=4] [N=3]	[T=3] [B=0] [R=1] [CB=1] [CT=5] [N=0]	[T=3] [B=0] [R=1] [CB=1] [CT=5] [N=1]	[T=3] [B=0] [R=1] [CB=1] [CT=5] [N=2]	[T=3] [B=0] [R=1] [CB=1] [CT=5] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=0] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=0] [N=1]	[T=3] [B=0] [R=1] [CB=2] [CT=0] [N=2]	[T=3] [B=0] [R=1] [CB=2] [CT=0] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=1] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=1] [N=1]
R111	ct_coef															
	[T=3] [B=0] [R=1] [CB=2] [CT=1] [N=2]	[T=3] [B=0] [R=1] [CB=2] [CT=1] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=2] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=2] [N=1]	[T=3] [B=0] [R=1] [CB=2] [CT=2] [N=2]	[T=3] [B=0] [R=1] [CB=2] [CT=2] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=3] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=3] [N=1]	[T=3] [B=0] [R=1] [CB=2] [CT=3] [N=2]	[T=3] [B=0] [R=1] [CB=2] [CT=3] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=4] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=4] [N=1]	[T=3] [B=0] [R=1] [CB=2] [CT=4] [N=2]	[T=3] [B=0] [R=1] [CB=2] [CT=4] [N=3]	[T=3] [B=0] [R=1] [CB=2] [CT=5] [N=0]	[T=3] [B=0] [R=1] [CB=2] [CT=5] [N=1]
R112	ct_coef															
	[T=3] [B=0] [R=1] [CB=2]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1]	[T=3] [B=0] [R=1] [CB=3]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CT=5] [N=2]	[CB=2] [CT=5] [N=3]	[CB=3] [CT=0] [N=0]	[CB=3] [CT=0] [N=1]	[CB=3] [CT=0] [N=2]	[CB=3] [CT=0] [N=3]	[CB=3] [CT=1] [N=0]	[CB=3] [CT=1] [N=1]	[CB=3] [CT=1] [N=2]	[CB=3] [CT=1] [N=3]	[CB=3] [CT=2] [N=0]	[CB=3] [CT=2] [N=1]	[CB=3] [CT=2] [N=2]	[CB=3] [CT=2] [N=3]	[CB=3] [CT=3] [N=0]	[CT=3] [N=1]
R113	ct_coef															
	[T=3] [B=0] [R=1] [CB=3] [CT=3] [N=2]	[T=3] [B=0] [R=1] [CB=3] [CT=3] [N=3]	[T=3] [B=0] [R=1] [CB=3] [CT=4] [N=0]	[T=3] [B=0] [R=1] [CB=3] [CT=4] [N=1]	[T=3] [B=0] [R=1] [CB=3] [CT=4] [N=2]	[T=3] [B=0] [R=1] [CB=3] [CT=4] [N=3]	[T=3] [B=0] [R=1] [CB=3] [CT=5] [N=0]	[T=3] [B=0] [R=1] [CB=3] [CT=5] [N=1]	[T=3] [B=0] [R=1] [CB=3] [CT=5] [N=2]	[T=3] [B=0] [R=1] [CB=3] [CT=5] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=0] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=0] [N=1]	[T=3] [B=0] [R=1] [CB=4] [CT=0] [N=2]	[T=3] [B=0] [R=1] [CB=4] [CT=0] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=1] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=1] [N=1]
R114	ct_coef															
	[T=3] [B=0] [R=1] [CB=4] [CT=1] [N=2]	[T=3] [B=0] [R=1] [CB=4] [CT=1] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=2] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=2] [N=1]	[T=3] [B=0] [R=1] [CB=4] [CT=2] [N=2]	[T=3] [B=0] [R=1] [CB=4] [CT=2] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=3] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=3] [N=1]	[T=3] [B=0] [R=1] [CB=4] [CT=3] [N=2]	[T=3] [B=0] [R=1] [CB=4] [CT=3] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=4] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=4] [N=1]	[T=3] [B=0] [R=1] [CB=4] [CT=4] [N=2]	[T=3] [B=0] [R=1] [CB=4] [CT=4] [N=3]	[T=3] [B=0] [R=1] [CB=4] [CT=5] [N=0]	[T=3] [B=0] [R=1] [CB=4] [CT=5] [N=1]
R115	ct_coef															
	[T=3] [B=0] [R=1] [CB=4] [CT=5] [N=2]	[T=3] [B=0] [R=1] [CB=4]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5] [CT=2]	[T=3] [B=0] [R=1] [CB=5] [CT=2]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5]	[T=3] [B=0] [R=1] [CB=5] [CT=3] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=5] [N=3]	[CT=0] [N=0]	[CT=0] [N=1]	[CT=0] [N=2]	[CT=0] [N=3]	[CT=1] [N=0]	[CT=1] [N=1]	[CT=1] [N=2]	[CT=1] [N=3]	[N=0]	[N=1]	[CT=2] [N=2]	[CT=2] [N=3]	[CT=3] [N=0]	
R116	ct_coef															
	[T=3] [B=0] [R=1] [CB=5] [CT=3] [N=2]	[T=3] [B=0] [R=1] [CB=5] [CT=3] [N=3]	[T=3] [B=0] [R=1] [CB=5] [CT=4] [N=0]	[T=3] [B=0] [R=1] [CB=5] [CT=4] [N=1]	[T=3] [B=0] [R=1] [CB=5] [CT=4] [N=2]	[T=3] [B=0] [R=1] [CB=5] [CT=4] [N=3]	[T=3] [B=0] [R=1] [CB=5] [CT=5] [N=0]	[T=3] [B=0] [R=1] [CB=5] [CT=5] [N=1]	[T=3] [B=0] [R=1] [CB=5] [CT=5] [N=2]	[T=3] [B=0] [R=1] [CB=5] [CT=5] [N=3]	[T=3] [B=1] [R=0] [CB=0] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=0] [CT=0] [N=1]	[T=3] [B=1] [R=0] [CB=0] [CT=0] [N=2]	[T=3] [B=1] [R=0] [CB=0] [CT=0] [N=3]	[T=3] [B=1] [R=0] [CB=0] [CT=1] [N=0]	[T=3] [B=1] [R=0] [CB=0] [CT=1] [N=1]
R117	ct_coef															
	[T=3] [B=1] [R=0] [CB=0] [CT=1] [N=2]	[T=3] [B=1] [R=0] [CB=0] [CT=1] [N=3]	[T=3] [B=1] [R=0] [CB=0] [CT=2] [N=0]	[T=3] [B=1] [R=0] [CB=0] [CT=2] [N=1]	[T=3] [B=1] [R=0] [CB=0] [CT=2] [N=2]	[T=3] [B=1] [R=0] [CB=0] [CT=2] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=0] [N=1]	[T=3] [B=1] [R=0] [CB=1] [CT=0] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=0] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=1] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=1] [N=1]	[T=3] [B=1] [R=0] [CB=1] [CT=1] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=1] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=2] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=2] [N=1]
R118	ct_coef															
	[T=3] [B=1] [R=0] [CB=1] [CT=2] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=2] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=3] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=3] [N=1]	[T=3] [B=1] [R=0] [CB=1] [CT=3] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=3] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=4] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=4] [N=1]	[T=3] [B=1] [R=0] [CB=1] [CT=4] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=4] [N=3]	[T=3] [B=1] [R=0] [CB=1] [CT=5] [N=0]	[T=3] [B=1] [R=0] [CB=1] [CT=5] [N=1]	[T=3] [B=1] [R=0] [CB=1] [CT=5] [N=2]	[T=3] [B=1] [R=0] [CB=1] [CT=5] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=0] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R119	ct_coef															
	[T=3] [B=1] [R=0] [CB=2] [CT=0] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=0] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=1] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=1] [N=1]	[T=3] [B=1] [R=0] [CB=2] [CT=1] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=1] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=2] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=2] [N=1]	[T=3] [B=1] [R=0] [CB=2] [CT=2] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=2] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=3] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=3] [N=1]	[T=3] [B=1] [R=0] [CB=2] [CT=3] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=3] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=4] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=4] [N=1]
R120	ct_coef															
	[T=3] [B=1] [R=0] [CB=2] [CT=4] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=4] [N=3]	[T=3] [B=1] [R=0] [CB=2] [CT=5] [N=0]	[T=3] [B=1] [R=0] [CB=2] [CT=5] [N=1]	[T=3] [B=1] [R=0] [CB=2] [CT=5] [N=2]	[T=3] [B=1] [R=0] [CB=2] [CT=5] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=0] [N=1]	[T=3] [B=1] [R=0] [CB=3] [CT=0] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=0] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=1] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=1] [N=1]	[T=3] [B=1] [R=0] [CB=3] [CT=1] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=1] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=2] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=2] [N=1]
R121	ct_coef															
	[T=3] [B=1] [R=0] [CB=3] [CT=2] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=2] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=3] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=3] [N=1]	[T=3] [B=1] [R=0] [CB=3] [CT=3] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=3] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=4] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=4] [N=1]	[T=3] [B=1] [R=0] [CB=3] [CT=4] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=4] [N=3]	[T=3] [B=1] [R=0] [CB=3] [CT=5] [N=0]	[T=3] [B=1] [R=0] [CB=3] [CT=5] [N=1]	[T=3] [B=1] [R=0] [CB=3] [CT=5] [N=2]	[T=3] [B=1] [R=0] [CB=3] [CT=5] [N=3]	[T=3] [B=1] [R=0] [CB=4] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=4] [CT=0] [N=1]
R122	ct_coef															
	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]	[T=3]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=1] [R=0] [CB=4] [CT=0] [N=2]	[B=1] [R=0] [CB=4] [CT=0] [N=3]	[B=1] [R=0] [CB=4] [CT=1] [N=0]	[B=1] [R=0] [CB=4] [CT=1] [N=1]	[B=1] [R=0] [CB=4] [CT=1] [N=2]	[B=1] [R=0] [CB=4] [CT=1] [N=3]	[B=1] [R=0] [CB=4] [CT=2] [N=0]	[B=1] [R=0] [CB=4] [CT=2] [N=1]	[B=1] [R=0] [CB=4] [CT=2] [N=2]	[B=1] [R=0] [CB=4] [CT=2] [N=3]	[B=1] [R=0] [CB=4] [CT=3] [N=0]	[B=1] [R=0] [CB=4] [CT=3] [N=1]	[B=1] [R=0] [CB=4] [CT=3] [N=2]	[B=1] [R=0] [CB=4] [CT=3] [N=3]	[B=1] [R=0] [CB=4] [CT=4] [N=0]	[B=1] [R=0] [CB=4] [CT=4] [N=1]
R123	ct_coef															
	[T=3] [B=1] [R=0] [CB=4] [CT=4] [N=2]	[T=3] [B=1] [R=0] [CB=4] [CT=4] [N=3]	[T=3] [B=1] [R=0] [CB=4] [CT=5] [N=0]	[T=3] [B=1] [R=0] [CB=4] [CT=5] [N=1]	[T=3] [B=1] [R=0] [CB=4] [CT=5] [N=2]	[T=3] [B=1] [R=0] [CB=4] [CT=5] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=0] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=0] [N=1]	[T=3] [B=1] [R=0] [CB=5] [CT=0] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=0] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=1] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=1] [N=1]	[T=3] [B=1] [R=0] [CB=5] [CT=1] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=1] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=2] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=2] [N=1]
R124	ct_coef															
	[T=3] [B=1] [R=0] [CB=5] [CT=2] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=2] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=3] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=3] [N=1]	[T=3] [B=1] [R=0] [CB=5] [CT=3] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=3] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=4] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=4] [N=1]	[T=3] [B=1] [R=0] [CB=5] [CT=4] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=4] [N=3]	[T=3] [B=1] [R=0] [CB=5] [CT=5] [N=0]	[T=3] [B=1] [R=0] [CB=5] [CT=5] [N=1]	[T=3] [B=1] [R=0] [CB=5] [CT=5] [N=2]	[T=3] [B=1] [R=0] [CB=5] [CT=5] [N=3]	[T=3] [B=1] [R=1] [CB=0] [CT=0] [N=0]	[T=3] [B=1] [R=1] [CB=0] [CT=0] [N=1]
R125	ct_coef															
	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]	[T=3] [B=1] [R=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=0] [CT=0] [N=2]	[CB=0] [CT=0] [N=3]	[CB=0] [CT=1] [N=0]	[CB=0] [CT=1] [N=1]	[CB=0] [CT=1] [N=2]	[CB=0] [CT=1] [N=3]	[CB=0] [CT=2] [N=0]	[CB=0] [CT=2] [N=1]	[CB=0] [CT=2] [N=2]	[CB=0] [CT=2] [N=3]	[CB=1] [CT=0] [N=0]	[CB=1] [CT=0] [N=1]	[CB=1] [CT=0] [N=2]	[CB=1] [CT=0] [N=3]	[CB=1] [CT=1] [N=0]	[CB=1] [CT=1] [N=1]
R126	ct_coef															
	[T=3] [B=1] [R=1] [CB=1] [CT=1] [N=2]	[T=3] [B=1] [R=1] [CB=1] [CT=1] [N=3]	[T=3] [B=1] [R=1] [CB=1] [CT=2] [N=0]	[T=3] [B=1] [R=1] [CB=1] [CT=2] [N=1]	[T=3] [B=1] [R=1] [CB=1] [CT=2] [N=2]	[T=3] [B=1] [R=1] [CB=1] [CT=2] [N=3]	[T=3] [B=1] [R=1] [CB=1] [CT=3] [N=0]	[T=3] [B=1] [R=1] [CB=1] [CT=3] [N=1]	[T=3] [B=1] [R=1] [CB=1] [CT=3] [N=2]	[T=3] [B=1] [R=1] [CB=1] [CT=3] [N=3]	[T=3] [B=1] [R=1] [CB=1] [CT=4] [N=0]	[T=3] [B=1] [R=1] [CB=1] [CT=4] [N=1]	[T=3] [B=1] [R=1] [CB=1] [CT=4] [N=2]	[T=3] [B=1] [R=1] [CB=1] [CT=4] [N=3]	[T=3] [B=1] [R=1] [CB=1] [CT=5] [N=0]	[T=3] [B=1] [R=1] [CB=1] [CT=5] [N=1]
R127	ct_coef															
	[T=3] [B=1] [R=1] [CB=1] [CT=5] [N=2]	[T=3] [B=1] [R=1] [CB=1] [CT=5] [N=3]	[T=3] [B=1] [R=1] [CB=2] [CT=0] [N=0]	[T=3] [B=1] [R=1] [CB=2] [CT=0] [N=1]	[T=3] [B=1] [R=1] [CB=2] [CT=0] [N=2]	[T=3] [B=1] [R=1] [CB=2] [CT=0] [N=3]	[T=3] [B=1] [R=1] [CB=2] [CT=1] [N=0]	[T=3] [B=1] [R=1] [CB=2] [CT=1] [N=1]	[T=3] [B=1] [R=1] [CB=2] [CT=1] [N=2]	[T=3] [B=1] [R=1] [CB=2] [CT=1] [N=3]	[T=3] [B=1] [R=1] [CB=2] [CT=2] [N=0]	[T=3] [B=1] [R=1] [CB=2] [CT=2] [N=1]	[T=3] [B=1] [R=1] [CB=2] [CT=2] [N=2]	[T=3] [B=1] [R=1] [CB=2] [CT=2] [N=3]	[T=3] [B=1] [R=1] [CB=2] [CT=3] [N=0]	[T=3] [B=1] [R=1] [CB=2] [CT=3] [N=1]
R128	ct_coef															
	[T=3] [B=1] [R=1] [CB=2] [CT=3]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=2]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3] [CT=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[N=2]	[CT=3] [N=3]	[CT=4] [N=0]	[CT=4] [N=1]	[CT=4] [N=2]	[CT=4] [N=3]	[CT=5] [N=0]	[CT=5] [N=1]	[CT=5] [N=2]	[CT=5] [N=3]	[CT=0] [N=0]	[CT=0] [N=1]	[CT=0] [N=2]	[CT=0] [N=3]	[CT=1] [N=0]	[N=1]
R129	ct_coef															
	[T=3] [B=1] [R=1] [CB=3] [CT=1] [N=2]	[T=3] [B=1] [R=1] [CB=3] [CT=1] [N=3]	[T=3] [B=1] [R=1] [CB=3] [CT=2] [N=0]	[T=3] [B=1] [R=1] [CB=3] [CT=2] [N=1]	[T=3] [B=1] [R=1] [CB=3] [CT=2] [N=2]	[T=3] [B=1] [R=1] [CB=3] [CT=2] [N=3]	[T=3] [B=1] [R=1] [CB=3] [CT=3] [N=0]	[T=3] [B=1] [R=1] [CB=3] [CT=3] [N=1]	[T=3] [B=1] [R=1] [CB=3] [CT=3] [N=2]	[T=3] [B=1] [R=1] [CB=3] [CT=3] [N=3]	[T=3] [B=1] [R=1] [CB=3] [CT=4] [N=0]	[T=3] [B=1] [R=1] [CB=3] [CT=4] [N=1]	[T=3] [B=1] [R=1] [CB=3] [CT=4] [N=2]	[T=3] [B=1] [R=1] [CB=3] [CT=4] [N=3]	[T=3] [B=1] [R=1] [CB=3] [CT=5] [N=0]	[T=3] [B=1] [R=1] [CB=3] [CT=5] [N=1]
R130	ct_coef															
	[T=3] [B=1] [R=1] [CB=3] [CT=5] [N=2]	[T=3] [B=1] [R=1] [CB=3] [CT=5] [N=3]	[T=3] [B=1] [R=1] [CB=4] [CT=0] [N=0]	[T=3] [B=1] [R=1] [CB=4] [CT=0] [N=1]	[T=3] [B=1] [R=1] [CB=4] [CT=0] [N=2]	[T=3] [B=1] [R=1] [CB=4] [CT=0] [N=3]	[T=3] [B=1] [R=1] [CB=4] [CT=1] [N=0]	[T=3] [B=1] [R=1] [CB=4] [CT=1] [N=1]	[T=3] [B=1] [R=1] [CB=4] [CT=1] [N=2]	[T=3] [B=1] [R=1] [CB=4] [CT=1] [N=3]	[T=3] [B=1] [R=1] [CB=4] [CT=2] [N=0]	[T=3] [B=1] [R=1] [CB=4] [CT=2] [N=1]	[T=3] [B=1] [R=1] [CB=4] [CT=2] [N=2]	[T=3] [B=1] [R=1] [CB=4] [CT=2] [N=3]	[T=3] [B=1] [R=1] [CB=4] [CT=3] [N=0]	[T=3] [B=1] [R=1] [CB=4] [CT=3] [N=1]
R131	ct_coef															
	[T=3] [B=1] [R=1] [CB=4] [CT=3] [N=2]	[T=3] [B=1] [R=1] [CB=4] [CT=3]	[T=3] [B=1] [R=1] [CB=4] [CT=4]	[T=3] [B=1] [R=1] [CB=4] [CT=4]	[T=3] [B=1] [R=1] [CB=4] [CT=4]	[T=3] [B=1] [R=1] [CB=4] [CT=4]	[T=3] [B=1] [R=1] [CB=4] [CT=5]	[T=3] [B=1] [R=1] [CB=4] [CT=5]	[T=3] [B=1] [R=1] [CB=4] [CT=5]	[T=3] [B=1] [R=1] [CB=4] [CT=5]	[T=3] [B=1] [R=1] [CB=5] [CT=0] [N=0]	[T=3] [B=1] [R=1] [CB=5] [CT=0] [N=1]	[T=3] [B=1] [R=1] [CB=5] [CT=0]	[T=3] [B=1] [R=1] [CB=5] [CT=0]	[T=3] [B=1] [R=1] [CB=5] [CT=1]	[T=3] [B=1] [R=1] [CB=5] [CT=1] [N=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[N=3]	[N=0]	[N=1]	[N=2]	[N=3]	[N=0]	[N=1]	[N=2]	[N=3]			[N=2]	[N=3]	[N=0]	
R132	ct_coef															
	[T=3] [B=1] [R=1] [CB=5] [CT=1] [N=2]	[T=3] [B=1] [R=1] [CB=5] [CT=1] [N=3]	[T=3] [B=1] [R=1] [CB=5] [CT=2] [N=0]	[T=3] [B=1] [R=1] [CB=5] [CT=2] [N=1]	[T=3] [B=1] [R=1] [CB=5] [CT=2] [N=2]	[T=3] [B=1] [R=1] [CB=5] [CT=2] [N=3]	[T=3] [B=1] [R=1] [CB=5] [CT=3] [N=0]	[T=3] [B=1] [R=1] [CB=5] [CT=3] [N=1]	[T=3] [B=1] [R=1] [CB=5] [CT=3] [N=2]	[T=3] [B=1] [R=1] [CB=5] [CT=3] [N=3]	[T=3] [B=1] [R=1] [CB=5] [CT=4] [N=0]	[T=3] [B=1] [R=1] [CB=5] [CT=4] [N=1]	[T=3] [B=1] [R=1] [CB=5] [CT=4] [N=2]	[T=3] [B=1] [R=1] [CB=5] [CT=4] [N=3]	[T=3] [B=1] [R=1] [CB=5] [CT=5] [N=0]	[T=3] [B=1] [R=1] [CB=5] [CT=5] [N=1]
R133	ct_coef		ct_mbskip						ct_inter_mode							
	[T=3] [B=1] [R=1] [CB=5] [CT=5] [N=2]	[T=3] [B=1] [R=1] [CB=5] [CT=5] [N=3]	[0] [0]	[0] [1]	[1] [0]	[1] [1]	[2] [0]	[2] [1]	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[1] [0]	[1] [1]	[1] [2]	[1] [3]
R134	ct_inter_mode															
	[2] [0]	[2] [1]	[2] [2]	[2] [3]	[3] [0]	[3] [1]	[3] [2]	[3] [3]	[4] [0]	[4] [1]	[4] [2]	[4] [3]	[5] [0]	[5] [1]	[5] [2]	[5] [3]
R135	ct_inter_mode				ct_switchable_interp											
	[6] [0]	[6] [1]	[6] [2]	[6] [3]	[0] [0]	[0] [1]	[0] [2]	[1] [0]	[1] [1]	[1] [2]	[2] [0]	[2] [1]	[2] [2]	[3] [0]	[3] [1]	[3] [2]
R136	ct_intra_inter								ct_comp_inter							
	[0] [0]	[0] [1]	[1] [0]	[1] [1]	[2] [0]	[2] [1]	[3] [0]	[3] [1]	[0] [0]	[0] [1]	[1] [0]	[1] [1]	[2] [0]	[2] [1]	[3] [0]	[3] [1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R137	ct_comp_inter		ct_single_ref													
	[4] [0]	[4] [1]	[0] [0] [0]	[0] [0] [1]	[0] [1] [0]	[0] [1] [1]	[1] [0] [0]	[1] [0] [1]	[1] [1] [0]	[1] [1] [1]	[2] [0] [0]	[2] [0] [1]	[2] [1] [0]	[2] [1] [1]	[3] [0] [0]	[3] [0] [1]
R138	ct_single_ref						ct_comp_ref									
	[3] [1] [0]	[3] [1] [1]	[4] [0] [0]	[4] [0] [1]	[4] [1] [0]	[4] [1] [1]	[0] [0]	[0] [1]	[1] [0]	[1] [1]	[2] [0]	[2] [1]	[3] [0]	[3] [1]	[4] [0]	[4] [1]
R139	ct_y_mode															
	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[0] [4]	[0] [5]	[0] [6]	[0] [7]	[0] [8]	[0] [9]	[1] [0]	[1] [1]	[1] [2]	[1] [3]	[1] [4]	[1] [5]
R140	ct_y_mode															
	[1] [6]	[1] [7]	[1] [8]	[1] [9]	[2] [0]	[2] [1]	[2] [2]	[2] [3]	[2] [4]	[2] [5]	[2] [6]	[2] [7]	[2] [8]	[2] [9]	[3] [0]	[3] [1]
R141	ct_y_mode								ct_partition							
	[3] [2]	[3] [3]	[3] [4]	[3] [5]	[3] [6]	[3] [7]	[3] [8]	[3] [9]	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[1] [0]	[1] [1]	[1] [2]	[1] [3]
R142	ct_partition															
	[2] [0]	[2] [1]	[2] [2]	[2] [3]	[3] [0]	[3] [1]	[3] [2]	[3] [3]	[4] [0]	[4] [1]	[4] [2]	[4] [3]	[5] [0]	[5] [1]	[5] [2]	[5] [3]
R143	ct_partition															
	[6] [0]	[6] [1]	[6] [2]	[6] [3]	[7] [0]	[7] [1]	[7] [2]	[7] [3]	[8] [0]	[8] [1]	[8] [2]	[8] [3]	[9] [0]	[9] [1]	[9] [2]	[9] [3]
R144	ct_partition															
	[10] [0]	[10] [1]	[10] [2]	[10] [3]	[11] [0]	[11] [1]	[11] [2]	[11] [3]	[12] [0]	[12] [1]	[12] [2]	[12] [3]	[13] [0]	[13] [1]	[13] [2]	[13] [3]
R145	ct_partition								ct_mv_joints				ct_mv_sign		ct_mv_classes	

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[14] [0]	[14] [1]	[14] [2]	[14] [3]	[15] [0]	[15] [1]	[15] [2]	[15] [3]	[0]	[1]	[2]	[3]	[0] [0]	[0] [1]	[0] [0]	[0] [1]
R146	ct_mv_classes									ct_mv_class0		ct_mv_bits				
	[0] [2]	[0] [3]	[0] [4]	[0] [5]	[0] [6]	[0] [7]	[0] [8]	[0] [9]	[0] [10]	[0] [0]	[0] [1]	[0] [0] [0]	[0] [0] [1]	[0] [1] [0]	[0] [1] [1]	[0] [2] [0]
R147	ct_mv_bits															ct_mv_sign
	[0] [2] [1]	[0] [3] [0]	[0] [3] [1]	[0] [4] [0]	[0] [4] [1]	[0] [5] [0]	[0] [5] [1]	[0] [6] [0]	[0] [6] [1]	[0] [7] [0]	[0] [7] [1]	[0] [8] [0]	[0] [8] [1]	[0] [9] [0]	[0] [9] [1]	[1] [0]
R148	ct_mv_sign	ct_mv_classes											ct_mv_class0		ct_mv_bits	
	[1] [1]	[1] [0]	[1] [1]	[1] [2]	[1] [3]	[1] [4]	[1] [5]	[1] [6]	[1] [7]	[1] [8]	[1] [9]	[1] [10]	[1] [0]	[1] [1]	[1] [0] [0]	[1] [0] [1]
R149	ct_mv_bits															
	[1] [1] [0]	[1] [1] [1]	[1] [2] [0]	[1] [2] [1]	[1] [3] [0]	[1] [3] [1]	[1] [4] [0]	[1] [4] [1]	[1] [5] [0]	[1] [5] [1]	[1] [6] [0]	[1] [6] [1]	[1] [7] [0]	[1] [7] [1]	[1] [8] [0]	[1] [8] [1]
R150	ct_mv_bits		ct_mv_class0_fp								ct_mv_fp				ct_mv_class0_fp	
	[1] [9] [0]	[1] [9] [1]	[0] [0] [0]	[0] [0] [1]	[0] [0] [2]	[0] [0] [3]	[0] [1] [0]	[0] [1] [1]	[0] [1] [2]	[0] [1] [3]	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[1] [0] [0]	[1] [0] [1]
R151	ct_mv_class0_fp						ct_mv_fp				ct_mv_class0_hp		ct_mv_hp		ct_mv_class0_hp	
	[1] [0] [2]	[1] [0] [3]	[1] [1] [0]	[1] [1] [1]	[1] [1] [2]	[1] [1] [3]	[1] [0]	[1] [1]	[1] [2]	[1] [3]	[0] [0]	[0] [1]	[0] [0]	[0] [1]	[1] [0]	[1] [1]
R152	ct_mv_hp		ct_eob_br													

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[1] [0]	[1] [1]	[T=0] [B=0] [R=0] [CB=0] [CT=0]	[T=0] [B=0] [R=0] [CB=0] [CT=1]	[T=0] [B=0] [R=0] [CB=0] [CT=2]	[T=0] [B=0] [R=0] [CB=1] [CT=0]	[T=0] [B=0] [R=0] [CB=1] [CT=1]	[T=0] [B=0] [R=0] [CB=1] [CT=2]	[T=0] [B=0] [R=0] [CB=1] [CT=3]	[T=0] [B=0] [R=0] [CB=1] [CT=4]	[T=0] [B=0] [R=0] [CB=1] [CT=5]	[T=0] [B=0] [R=0] [CB=2] [CT=0]	[T=0] [B=0] [R=0] [CB=2] [CT=1]	[T=0] [B=0] [R=0] [CB=2] [CT=2]	[T=0] [B=0] [R=0] [CB=2] [CT=3]	[T=0] [B=0] [R=0] [CB=2] [CT=4]
R153	ct_eob_br															
	[T=0] [B=0] [R=0] [CB=2] [CT=5]	[T=0] [B=0] [R=0] [CB=3] [CT=0]	[T=0] [B=0] [R=0] [CB=3] [CT=1]	[T=0] [B=0] [R=0] [CB=3] [CT=2]	[T=0] [B=0] [R=0] [CB=3] [CT=3]	[T=0] [B=0] [R=0] [CB=3] [CT=4]	[T=0] [B=0] [R=0] [CB=3] [CT=5]	[T=0] [B=0] [R=0] [CB=4] [CT=0]	[T=0] [B=0] [R=0] [CB=4] [CT=1]	[T=0] [B=0] [R=0] [CB=4] [CT=2]	[T=0] [B=0] [R=0] [CB=4] [CT=3]	[T=0] [B=0] [R=0] [CB=4] [CT=4]	[T=0] [B=0] [R=0] [CB=4] [CT=5]	[T=0] [B=0] [R=0] [CB=5] [CT=0]	[T=0] [B=0] [R=0] [CB=5] [CT=1]	[T=0] [B=0] [R=0] [CB=5] [CT=2]
R154	ct_eob_br															
	[T=0] [B=0] [R=0] [CB=5] [CT=3]	[T=0] [B=0] [R=0] [CB=5] [CT=4]	[T=0] [B=0] [R=0] [CB=5] [CT=5]	[T=0] [B=0] [R=1] [CB=0] [CT=0]	[T=0] [B=0] [R=1] [CB=0] [CT=1]	[T=0] [B=0] [R=1] [CB=0] [CT=2]	[T=0] [B=0] [R=1] [CB=1] [CT=0]	[T=0] [B=0] [R=1] [CB=1] [CT=1]	[T=0] [B=0] [R=1] [CB=1] [CT=2]	[T=0] [B=0] [R=1] [CB=1] [CT=3]	[T=0] [B=0] [R=1] [CB=1] [CT=4]	[T=0] [B=0] [R=1] [CB=1] [CT=5]	[T=0] [B=0] [R=1] [CB=2] [CT=0]	[T=0] [B=0] [R=1] [CB=2] [CT=1]	[T=0] [B=0] [R=1] [CB=2] [CT=2]	[T=0] [B=0] [R=1] [CB=2] [CT=3]
R155	ct_eob_br															
	[T=0] [B=0] [R=1] [CB=2] [CT=4]	[T=0] [B=0] [R=1] [CB=2]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=3]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=4]	[T=0] [B=0] [R=1] [CB=5]	[T=0] [B=0] [R=1] [CB=5] [CT=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=5]	[CT=0]	[CT=1]	[CT=2]	[CT=3]	[CT=4]	[CT=5]	[CT=0]	[CT=1]	[CT=2]	[CT=3]	[CT=4]	[CT=5]	[CT=0]	
R156	ct_eob_br															
	[T=0] [B=0] [R=1] [CB=5] [CT=2]	[T=0] [B=0] [R=1] [CB=5] [CT=3]	[T=0] [B=0] [R=1] [CB=5] [CT=4]	[T=0] [B=0] [R=1] [CB=5] [CT=5]	[T=0] [B=1] [R=0] [CB=0] [CT=0]	[T=0] [B=1] [R=0] [CB=0] [CT=1]	[T=0] [B=1] [R=0] [CB=0] [CT=2]	[T=0] [B=1] [R=0] [CB=1] [CT=0]	[T=0] [B=1] [R=0] [CB=1] [CT=1]	[T=0] [B=1] [R=0] [CB=1] [CT=2]	[T=0] [B=1] [R=0] [CB=1] [CT=3]	[T=0] [B=1] [R=0] [CB=1] [CT=4]	[T=0] [B=1] [R=0] [CB=1] [CT=5]	[T=0] [B=1] [R=0] [CB=2] [CT=0]	[T=0] [B=1] [R=0] [CB=2] [CT=1]	[T=0] [B=1] [R=0] [CB=2] [CT=2]
R157	ct_eob_br															
	[T=0] [B=1] [R=0] [CB=2] [CT=3]	[T=0] [B=1] [R=0] [CB=2] [CT=4]	[T=0] [B=1] [R=0] [CB=2] [CT=5]	[T=0] [B=1] [R=0] [CB=3] [CT=0]	[T=0] [B=1] [R=0] [CB=3] [CT=1]	[T=0] [B=1] [R=0] [CB=3] [CT=2]	[T=0] [B=1] [R=0] [CB=3] [CT=3]	[T=0] [B=1] [R=0] [CB=3] [CT=4]	[T=0] [B=1] [R=0] [CB=3] [CT=5]	[T=0] [B=1] [R=0] [CB=4] [CT=0]	[T=0] [B=1] [R=0] [CB=4] [CT=1]	[T=0] [B=1] [R=0] [CB=4] [CT=2]	[T=0] [B=1] [R=0] [CB=4] [CT=3]	[T=0] [B=1] [R=0] [CB=4] [CT=4]	[T=0] [B=1] [R=0] [CB=4] [CT=5]	[T=0] [B=1] [R=0] [CB=5] [CT=0]
R158	ct_eob_br															
	[T=0] [B=1] [R=0] [CB=5] [CT=1]	[T=0] [B=1] [R=0] [CB=5] [CT=2]	[T=0] [B=1] [R=0] [CB=5] [CT=3]	[T=0] [B=1] [R=0] [CB=5] [CT=4]	[T=0] [B=1] [R=0] [CB=5] [CT=5]	[T=0] [B=1] [R=1] [CB=0] [CT=0]	[T=0] [B=1] [R=1] [CB=0] [CT=1]	[T=0] [B=1] [R=1] [CB=0] [CT=2]	[T=0] [B=1] [R=1] [CB=1] [CT=0]	[T=0] [B=1] [R=1] [CB=1] [CT=1]	[T=0] [B=1] [R=1] [CB=1] [CT=2]	[T=0] [B=1] [R=1] [CB=1] [CT=3]	[T=0] [B=1] [R=1] [CB=1] [CT=4]	[T=0] [B=1] [R=1] [CB=1] [CT=5]	[T=0] [B=1] [R=1] [CB=2] [CT=0]	[T=0] [B=1] [R=1] [CB=2] [CT=1]
R159	ct_eob_br															
	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]	[T=0] [B=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[R=1] [CB=2] [CT=2]	[R=1] [CB=2] [CT=3]	[R=1] [CB=2] [CT=4]	[R=1] [CB=2] [CT=5]	[R=1] [CB=3] [CT=0]	[R=1] [CB=3] [CT=1]	[R=1] [CB=3] [CT=2]	[R=1] [CB=3] [CT=3]	[R=1] [CB=3] [CT=4]	[R=1] [CB=3] [CT=5]	[R=1] [CB=4] [CT=0]	[R=1] [CB=4] [CT=1]	[R=1] [CB=4] [CT=2]	[R=1] [CB=4] [CT=3]	[R=1] [CB=4] [CT=4]	[R=1] [CB=4] [CT=5]
R160	ct_eob_br															
	[T=0] [B=1] [R=1] [CB=5] [CT=0]	[T=0] [B=1] [R=1] [CB=5] [CT=1]	[T=0] [B=1] [R=1] [CB=5] [CT=2]	[T=0] [B=1] [R=1] [CB=5] [CT=3]	[T=0] [B=1] [R=1] [CB=5] [CT=4]	[T=0] [B=1] [R=1] [CB=5] [CT=5]	[T=1] [B=0] [R=0] [CB=0] [CT=0]	[T=1] [B=0] [R=0] [CB=0] [CT=1]	[T=1] [B=0] [R=0] [CB=0] [CT=2]	[T=1] [B=0] [R=0] [CB=1] [CT=0]	[T=1] [B=0] [R=0] [CB=1] [CT=1]	[T=1] [B=0] [R=0] [CB=1] [CT=2]	[T=1] [B=0] [R=0] [CB=1] [CT=3]	[T=1] [B=0] [R=0] [CB=1] [CT=4]	[T=1] [B=0] [R=0] [CB=1] [CT=5]	[T=1] [B=0] [R=0] [CB=2] [CT=0]
R161	ct_eob_br															
	[T=1] [B=0] [R=0] [CB=2] [CT=1]	[T=1] [B=0] [R=0] [CB=2] [CT=2]	[T=1] [B=0] [R=0] [CB=2] [CT=3]	[T=1] [B=0] [R=0] [CB=2] [CT=4]	[T=1] [B=0] [R=0] [CB=2] [CT=5]	[T=1] [B=0] [R=0] [CB=3] [CT=0]	[T=1] [B=0] [R=0] [CB=3] [CT=1]	[T=1] [B=0] [R=0] [CB=3] [CT=2]	[T=1] [B=0] [R=0] [CB=3] [CT=3]	[T=1] [B=0] [R=0] [CB=3] [CT=4]	[T=1] [B=0] [R=0] [CB=3] [CT=5]	[T=1] [B=0] [R=0] [CB=4] [CT=0]	[T=1] [B=0] [R=0] [CB=4] [CT=1]	[T=1] [B=0] [R=0] [CB=4] [CT=2]	[T=1] [B=0] [R=0] [CB=4] [CT=3]	[T=1] [B=0] [R=0] [CB=4] [CT=4]
R162	ct_eob_br															
	[T=1] [B=0] [R=0] [CB=4] [CT=5]	[T=1] [B=0] [R=0] [CB=5] [CT=0]	[T=1] [B=0] [R=0] [CB=5] [CT=1]	[T=1] [B=0] [R=0] [CB=5] [CT=2]	[T=1] [B=0] [R=0] [CB=5] [CT=3]	[T=1] [B=0] [R=0] [CB=5] [CT=4]	[T=1] [B=0] [R=0] [CB=5] [CT=5]	[T=1] [B=0] [R=1] [CB=0] [CT=0]	[T=1] [B=0] [R=1] [CB=0] [CT=1]	[T=1] [B=0] [R=1] [CB=0] [CT=2]	[T=1] [B=0] [R=1] [CB=1] [CT=0]	[T=1] [B=0] [R=1] [CB=1] [CT=1]	[T=1] [B=0] [R=1] [CB=1] [CT=2]	[T=1] [B=0] [R=1] [CB=1] [CT=3]	[T=1] [B=0] [R=1] [CB=1] [CT=4]	[T=1] [B=0] [R=1] [CB=1] [CT=5]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R163	ct_eob_br															
	[T=1] [B=0] [R=1] [CB=2] [CT=0]	[T=1] [B=0] [R=1] [CB=2] [CT=1]	[T=1] [B=0] [R=1] [CB=2] [CT=2]	[T=1] [B=0] [R=1] [CB=2] [CT=3]	[T=1] [B=0] [R=1] [CB=2] [CT=4]	[T=1] [B=0] [R=1] [CB=2] [CT=5]	[T=1] [B=0] [R=1] [CB=3] [CT=0]	[T=1] [B=0] [R=1] [CB=3] [CT=1]	[T=1] [B=0] [R=1] [CB=3] [CT=2]	[T=1] [B=0] [R=1] [CB=3] [CT=3]	[T=1] [B=0] [R=1] [CB=3] [CT=4]	[T=1] [B=0] [R=1] [CB=3] [CT=5]	[T=1] [B=0] [R=1] [CB=4] [CT=0]	[T=1] [B=0] [R=1] [CB=4] [CT=1]	[T=1] [B=0] [R=1] [CB=4] [CT=2]	[T=1] [B=0] [R=1] [CB=4] [CT=3]
R164	ct_eob_br															
	[T=1] [B=0] [R=1] [CB=4] [CT=4]	[T=1] [B=0] [R=1] [CB=4] [CT=5]	[T=1] [B=0] [R=1] [CB=5] [CT=0]	[T=1] [B=0] [R=1] [CB=5] [CT=1]	[T=1] [B=0] [R=1] [CB=5] [CT=2]	[T=1] [B=0] [R=1] [CB=5] [CT=3]	[T=1] [B=0] [R=1] [CB=5] [CT=4]	[T=1] [B=0] [R=1] [CB=5] [CT=5]	[T=1] [B=1] [R=0] [CB=0] [CT=0]	[T=1] [B=1] [R=0] [CB=0] [CT=1]	[T=1] [B=1] [R=0] [CB=0] [CT=2]	[T=1] [B=1] [R=0] [CB=1] [CT=0]	[T=1] [B=1] [R=0] [CB=1] [CT=1]	[T=1] [B=1] [R=0] [CB=1] [CT=2]	[T=1] [B=1] [R=0] [CB=1] [CT=3]	[T=1] [B=1] [R=0] [CB=1] [CT=4]
R165	ct_eob_br															
	[T=1] [B=1] [R=0] [CB=1] [CT=5]	[T=1] [B=1] [R=0] [CB=2] [CT=0]	[T=1] [B=1] [R=0] [CB=2] [CT=1]	[T=1] [B=1] [R=0] [CB=2] [CT=2]	[T=1] [B=1] [R=0] [CB=2] [CT=3]	[T=1] [B=1] [R=0] [CB=2] [CT=4]	[T=1] [B=1] [R=0] [CB=2] [CT=5]	[T=1] [B=1] [R=0] [CB=3] [CT=0]	[T=1] [B=1] [R=0] [CB=3] [CT=1]	[T=1] [B=1] [R=0] [CB=3] [CT=2]	[T=1] [B=1] [R=0] [CB=3] [CT=3]	[T=1] [B=1] [R=0] [CB=3] [CT=4]	[T=1] [B=1] [R=0] [CB=3] [CT=5]	[T=1] [B=1] [R=0] [CB=4] [CT=0]	[T=1] [B=1] [R=0] [CB=4] [CT=1]	[T=1] [B=1] [R=0] [CB=4] [CT=2]
R166	ct_eob_br															
	[T=1] [B=1] [R=0] [CB=4]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=0]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1]	[T=1] [B=1] [R=1] [CB=1]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CT=3]	[CB=4] [CT=4]	[CB=4] [CT=5]	[CB=5] [CT=0]	[CB=5] [CT=1]	[CB=5] [CT=2]	[CB=5] [CT=3]	[CB=5] [CT=4]	[CB=5] [CT=5]	[CB=0] [CT=0]	[CB=0] [CT=1]	[CB=0] [CT=2]	[CB=1] [CT=0]	[CB=1] [CT=1]	[CB=1] [CT=2]	[CT=3]
R167	ct_eob_br															
	[T=1] [B=1] [R=1] [CB=1] [CT=4]	[T=1] [B=1] [R=1] [CB=1] [CT=5]	[T=1] [B=1] [R=1] [CB=2] [CT=0]	[T=1] [B=1] [R=1] [CB=2] [CT=1]	[T=1] [B=1] [R=1] [CB=2] [CT=2]	[T=1] [B=1] [R=1] [CB=2] [CT=3]	[T=1] [B=1] [R=1] [CB=2] [CT=4]	[T=1] [B=1] [R=1] [CB=2] [CT=5]	[T=1] [B=1] [R=1] [CB=3] [CT=0]	[T=1] [B=1] [R=1] [CB=3] [CT=1]	[T=1] [B=1] [R=1] [CB=3] [CT=2]	[T=1] [B=1] [R=1] [CB=3] [CT=3]	[T=1] [B=1] [R=1] [CB=3] [CT=4]	[T=1] [B=1] [R=1] [CB=3] [CT=5]	[T=1] [B=1] [R=1] [CB=4] [CT=0]	[T=1] [B=1] [R=1] [CB=4] [CT=1]
R168	ct_eob_br															
	[T=1] [B=1] [R=1] [CB=4] [CT=2]	[T=1] [B=1] [R=1] [CB=4] [CT=3]	[T=1] [B=1] [R=1] [CB=4] [CT=4]	[T=1] [B=1] [R=1] [CB=4] [CT=5]	[T=1] [B=1] [R=1] [CB=5] [CT=0]	[T=1] [B=1] [R=1] [CB=5] [CT=1]	[T=1] [B=1] [R=1] [CB=5] [CT=2]	[T=1] [B=1] [R=1] [CB=5] [CT=3]	[T=1] [B=1] [R=1] [CB=5] [CT=4]	[T=1] [B=1] [R=1] [CB=5] [CT=5]	[T=2] [B=0] [R=0] [CB=0] [CT=0]	[T=2] [B=0] [R=0] [CB=0] [CT=1]	[T=2] [B=0] [R=0] [CB=0] [CT=2]	[T=2] [B=0] [R=0] [CB=1] [CT=0]	[T=2] [B=0] [R=0] [CB=1] [CT=1]	[T=2] [B=0] [R=0] [CB=1] [CT=2]
R169	ct_eob_br															
	[T=2] [B=0] [R=0] [CB=1] [CT=3]	[T=2] [B=0] [R=0] [CB=1] [CT=4]	[T=2] [B=0] [R=0] [CB=1] [CT=5]	[T=2] [B=0] [R=0] [CB=2] [CT=0]	[T=2] [B=0] [R=0] [CB=2] [CT=1]	[T=2] [B=0] [R=0] [CB=2] [CT=2]	[T=2] [B=0] [R=0] [CB=2] [CT=3]	[T=2] [B=0] [R=0] [CB=2] [CT=4]	[T=2] [B=0] [R=0] [CB=2] [CT=5]	[T=2] [B=0] [R=0] [CB=3] [CT=0]	[T=2] [B=0] [R=0] [CB=3] [CT=1]	[T=2] [B=0] [R=0] [CB=3] [CT=2]	[T=2] [B=0] [R=0] [CB=3] [CT=3]	[T=2] [B=0] [R=0] [CB=3] [CT=4]	[T=2] [B=0] [R=0] [CB=3] [CT=5]	[T=2] [B=0] [R=0] [CB=4] [CT=0]
R170	ct_eob_br															
	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]	[T=2]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[B=0] [R=0] [CB=4] [CT=1]	[B=0] [R=0] [CB=4] [CT=2]	[B=0] [R=0] [CB=4] [CT=3]	[B=0] [R=0] [CB=4] [CT=4]	[B=0] [R=0] [CB=4] [CT=5]	[B=0] [R=0] [CB=5] [CT=0]	[B=0] [R=0] [CB=5] [CT=1]	[B=0] [R=0] [CB=5] [CT=2]	[B=0] [R=0] [CB=5] [CT=3]	[B=0] [R=0] [CB=5] [CT=4]	[B=0] [R=0] [CB=5] [CT=5]	[B=0] [R=1] [CB=0] [CT=0]	[B=0] [R=1] [CB=0] [CT=1]	[B=0] [R=1] [CB=0] [CT=2]	[B=0] [R=1] [CB=1] [CT=0]	[B=0] [R=1] [CB=1] [CT=1]
R171	ct_eob_br															
	[T=2] [B=0] [R=1] [CB=1] [CT=2]	[T=2] [B=0] [R=1] [CB=1] [CT=3]	[T=2] [B=0] [R=1] [CB=1] [CT=4]	[T=2] [B=0] [R=1] [CB=1] [CT=5]	[T=2] [B=0] [R=1] [CB=2] [CT=0]	[T=2] [B=0] [R=1] [CB=2] [CT=1]	[T=2] [B=0] [R=1] [CB=2] [CT=2]	[T=2] [B=0] [R=1] [CB=2] [CT=3]	[T=2] [B=0] [R=1] [CB=2] [CT=4]	[T=2] [B=0] [R=1] [CB=2] [CT=5]	[T=2] [B=0] [R=1] [CB=3] [CT=0]	[T=2] [B=0] [R=1] [CB=3] [CT=1]	[T=2] [B=0] [R=1] [CB=3] [CT=2]	[T=2] [B=0] [R=1] [CB=3] [CT=3]	[T=2] [B=0] [R=1] [CB=3] [CT=4]	[T=2] [B=0] [R=1] [CB=3] [CT=5]
R172	ct_eob_br															
	[T=2] [B=0] [R=1] [CB=4] [CT=0]	[T=2] [B=0] [R=1] [CB=4] [CT=1]	[T=2] [B=0] [R=1] [CB=4] [CT=2]	[T=2] [B=0] [R=1] [CB=4] [CT=3]	[T=2] [B=0] [R=1] [CB=4] [CT=4]	[T=2] [B=0] [R=1] [CB=4] [CT=5]	[T=2] [B=0] [R=1] [CB=5] [CT=0]	[T=2] [B=0] [R=1] [CB=5] [CT=1]	[T=2] [B=0] [R=1] [CB=5] [CT=2]	[T=2] [B=0] [R=1] [CB=5] [CT=3]	[T=2] [B=0] [R=1] [CB=5] [CT=4]	[T=2] [B=0] [R=1] [CB=5] [CT=5]	[T=2] [B=1] [R=0] [CB=0] [CT=0]	[T=2] [B=1] [R=0] [CB=0] [CT=1]	[T=2] [B=1] [R=0] [CB=0] [CT=2]	[T=2] [B=1] [R=0] [CB=1] [CT=0]
R173	ct_eob_br															
	[T=2] [B=1] [R=0] [CB=1] [CT=1]	[T=2] [B=1] [R=0] [CB=1]	[T=2] [B=1] [R=0] [CB=1]	[T=2] [B=1] [R=0] [CB=1]	[T=2] [B=1] [R=0] [CB=1]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2]	[T=2] [B=1] [R=0] [CB=2] [CT=5]	[T=2] [B=1] [R=0] [CB=3] [CT=0]	[T=2] [B=1] [R=0] [CB=3]	[T=2] [B=1] [R=0] [CB=3]	[T=2] [B=1] [R=0] [CB=3]	[T=2] [B=1] [R=0] [CB=3] [CT=4]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=2]	[CT=3]	[CT=4]	[CT=5]	[CT=0]	[CT=1]	[CT=2]	[CT=3]	[CT=4]			[CT=1]	[CT=2]	[CT=3]	
R174	ct_eob_br															
	[T=2] [B=1] [R=0] [CB=3] [CT=5]	[T=2] [B=1] [R=0] [CB=4] [CT=0]	[T=2] [B=1] [R=0] [CB=4] [CT=1]	[T=2] [B=1] [R=0] [CB=4] [CT=2]	[T=2] [B=1] [R=0] [CB=4] [CT=3]	[T=2] [B=1] [R=0] [CB=4] [CT=4]	[T=2] [B=1] [R=0] [CB=4] [CT=5]	[T=2] [B=1] [R=0] [CB=5] [CT=0]	[T=2] [B=1] [R=0] [CB=5] [CT=1]	[T=2] [B=1] [R=0] [CB=5] [CT=2]	[T=2] [B=1] [R=0] [CB=5] [CT=3]	[T=2] [B=1] [R=0] [CB=5] [CT=4]	[T=2] [B=1] [R=0] [CB=5] [CT=5]	[T=2] [B=1] [R=1] [CB=0] [CT=0]	[T=2] [B=1] [R=1] [CB=0] [CT=1]	[T=2] [B=1] [R=1] [CB=0] [CT=2]
R175	ct_eob_br															
	[T=2] [B=1] [R=1] [CB=1] [CT=0]	[T=2] [B=1] [R=1] [CB=1] [CT=1]	[T=2] [B=1] [R=1] [CB=1] [CT=2]	[T=2] [B=1] [R=1] [CB=1] [CT=3]	[T=2] [B=1] [R=1] [CB=1] [CT=4]	[T=2] [B=1] [R=1] [CB=1] [CT=5]	[T=2] [B=1] [R=1] [CB=2] [CT=0]	[T=2] [B=1] [R=1] [CB=2] [CT=1]	[T=2] [B=1] [R=1] [CB=2] [CT=2]	[T=2] [B=1] [R=1] [CB=2] [CT=3]	[T=2] [B=1] [R=1] [CB=2] [CT=4]	[T=2] [B=1] [R=1] [CB=2] [CT=5]	[T=2] [B=1] [R=1] [CB=3] [CT=0]	[T=2] [B=1] [R=1] [CB=3] [CT=1]	[T=2] [B=1] [R=1] [CB=3] [CT=2]	[T=2] [B=1] [R=1] [CB=3] [CT=3]
R176	ct_eob_br															
	[T=2] [B=1] [R=1] [CB=3] [CT=4]	[T=2] [B=1] [R=1] [CB=3] [CT=5]	[T=2] [B=1] [R=1] [CB=4] [CT=0]	[T=2] [B=1] [R=1] [CB=4] [CT=1]	[T=2] [B=1] [R=1] [CB=4] [CT=2]	[T=2] [B=1] [R=1] [CB=4] [CT=3]	[T=2] [B=1] [R=1] [CB=4] [CT=4]	[T=2] [B=1] [R=1] [CB=4] [CT=5]	[T=2] [B=1] [R=1] [CB=5] [CT=0]	[T=2] [B=1] [R=1] [CB=5] [CT=1]	[T=2] [B=1] [R=1] [CB=5] [CT=2]	[T=2] [B=1] [R=1] [CB=5] [CT=3]	[T=2] [B=1] [R=1] [CB=5] [CT=4]	[T=2] [B=1] [R=1] [CB=5] [CT=5]	[T=3] [B=0] [R=0] [CB=0] [CT=0]	[T=3] [B=0] [R=0] [CB=0] [CT=1]
R177	ct_eob_br															
	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]	[T=3] [B=0] [R=0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[CB=0] [CT=2]	[CB=1] [CT=0]	[CB=1] [CT=1]	[CB=1] [CT=2]	[CB=1] [CT=3]	[CB=1] [CT=4]	[CB=1] [CT=5]	[CB=2] [CT=0]	[CB=2] [CT=1]	[CB=2] [CT=2]	[CB=2] [CT=3]	[CB=2] [CT=4]	[CB=2] [CT=5]	[CB=3] [CT=0]	[CB=3] [CT=1]	[CB=3] [CT=2]
R178	ct_eob_br															
	[T=3] [B=0] [R=0] [CB=3] [CT=3]	[T=3] [B=0] [R=0] [CB=3] [CT=4]	[T=3] [B=0] [R=0] [CB=3] [CT=5]	[T=3] [B=0] [R=0] [CB=4] [CT=0]	[T=3] [B=0] [R=0] [CB=4] [CT=1]	[T=3] [B=0] [R=0] [CB=4] [CT=2]	[T=3] [B=0] [R=0] [CB=4] [CT=3]	[T=3] [B=0] [R=0] [CB=4] [CT=4]	[T=3] [B=0] [R=0] [CB=4] [CT=5]	[T=3] [B=0] [R=0] [CB=5] [CT=0]	[T=3] [B=0] [R=0] [CB=5] [CT=1]	[T=3] [B=0] [R=0] [CB=5] [CT=2]	[T=3] [B=0] [R=0] [CB=5] [CT=3]	[T=3] [B=0] [R=0] [CB=5] [CT=4]	[T=3] [B=0] [R=0] [CB=5] [CT=5]	[T=3] [B=0] [R=1] [CB=0] [CT=0]
R179	ct_eob_br															
	[T=3] [B=0] [R=1] [CB=0] [CT=1]	[T=3] [B=0] [R=1] [CB=0] [CT=2]	[T=3] [B=0] [R=1] [CB=1] [CT=0]	[T=3] [B=0] [R=1] [CB=1] [CT=1]	[T=3] [B=0] [R=1] [CB=1] [CT=2]	[T=3] [B=0] [R=1] [CB=1] [CT=3]	[T=3] [B=0] [R=1] [CB=1] [CT=4]	[T=3] [B=0] [R=1] [CB=1] [CT=5]	[T=3] [B=0] [R=1] [CB=2] [CT=0]	[T=3] [B=0] [R=1] [CB=2] [CT=1]	[T=3] [B=0] [R=1] [CB=2] [CT=2]	[T=3] [B=0] [R=1] [CB=2] [CT=3]	[T=3] [B=0] [R=1] [CB=2] [CT=4]	[T=3] [B=0] [R=1] [CB=2] [CT=5]	[T=3] [B=0] [R=1] [CB=3] [CT=0]	[T=3] [B=0] [R=1] [CB=3] [CT=1]
R180	ct_eob_br															
	[T=3] [B=0] [R=1] [CB=3] [CT=2]	[T=3] [B=0] [R=1] [CB=3] [CT=3]	[T=3] [B=0] [R=1] [CB=3] [CT=4]	[T=3] [B=0] [R=1] [CB=3] [CT=5]	[T=3] [B=0] [R=1] [CB=4] [CT=0]	[T=3] [B=0] [R=1] [CB=4] [CT=1]	[T=3] [B=0] [R=1] [CB=4] [CT=2]	[T=3] [B=0] [R=1] [CB=4] [CT=3]	[T=3] [B=0] [R=1] [CB=4] [CT=4]	[T=3] [B=0] [R=1] [CB=4] [CT=5]	[T=3] [B=0] [R=1] [CB=5] [CT=0]	[T=3] [B=0] [R=1] [CB=5] [CT=1]	[T=3] [B=0] [R=1] [CB=5] [CT=2]	[T=3] [B=0] [R=1] [CB=5] [CT=3]	[T=3] [B=0] [R=1] [CB=5] [CT=4]	[T=3] [B=0] [R=1] [CB=5] [CT=5]
R181	ct_eob_br															

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[T=3] [B=1] [R=0] [CB=0] [CT=0]	[T=3] [B=1] [R=0] [CB=0] [CT=1]	[T=3] [B=1] [R=0] [CB=0] [CT=2]	[T=3] [B=1] [R=0] [CB=1] [CT=0]	[T=3] [B=1] [R=0] [CB=1] [CT=1]	[T=3] [B=1] [R=0] [CB=1] [CT=2]	[T=3] [B=1] [R=0] [CB=1] [CT=3]	[T=3] [B=1] [R=0] [CB=1] [CT=4]	[T=3] [B=1] [R=0] [CB=1] [CT=5]	[T=3] [B=1] [R=0] [CB=2] [CT=0]	[T=3] [B=1] [R=0] [CB=2] [CT=1]	[T=3] [B=1] [R=0] [CB=2] [CT=2]	[T=3] [B=1] [R=0] [CB=2] [CT=3]	[T=3] [B=1] [R=0] [CB=2] [CT=4]	[T=3] [B=1] [R=0] [CB=2] [CT=5]	[T=3] [B=1] [R=0] [CB=3] [CT=0]
R182	ct_eob_br															
	[T=3] [B=1] [R=0] [CB=3] [CT=1]	[T=3] [B=1] [R=0] [CB=3] [CT=2]	[T=3] [B=1] [R=0] [CB=3] [CT=3]	[T=3] [B=1] [R=0] [CB=3] [CT=4]	[T=3] [B=1] [R=0] [CB=3] [CT=5]	[T=3] [B=1] [R=0] [CB=4] [CT=0]	[T=3] [B=1] [R=0] [CB=4] [CT=1]	[T=3] [B=1] [R=0] [CB=4] [CT=2]	[T=3] [B=1] [R=0] [CB=4] [CT=3]	[T=3] [B=1] [R=0] [CB=4] [CT=4]	[T=3] [B=1] [R=0] [CB=4] [CT=5]	[T=3] [B=1] [R=0] [CB=5] [CT=0]	[T=3] [B=1] [R=0] [CB=5] [CT=1]	[T=3] [B=1] [R=0] [CB=5] [CT=2]	[T=3] [B=1] [R=0] [CB=5] [CT=3]	[T=3] [B=1] [R=0] [CB=5] [CT=4]
R183	ct_eob_br															
	[T=3] [B=1] [R=0] [CB=5] [CT=5]	[T=3] [B=1] [R=1] [CB=0] [CT=0]	[T=3] [B=1] [R=1] [CB=0] [CT=1]	[T=3] [B=1] [R=1] [CB=0] [CT=2]	[T=3] [B=1] [R=1] [CB=1] [CT=0]	[T=3] [B=1] [R=1] [CB=1] [CT=1]	[T=3] [B=1] [R=1] [CB=1] [CT=2]	[T=3] [B=1] [R=1] [CB=1] [CT=3]	[T=3] [B=1] [R=1] [CB=1] [CT=4]	[T=3] [B=1] [R=1] [CB=1] [CT=5]	[T=3] [B=1] [R=1] [CB=2] [CT=0]	[T=3] [B=1] [R=1] [CB=2] [CT=1]	[T=3] [B=1] [R=1] [CB=2] [CT=2]	[T=3] [B=1] [R=1] [CB=2] [CT=3]	[T=3] [B=1] [R=1] [CB=2] [CT=4]	[T=3] [B=1] [R=1] [CB=2] [CT=5]
R184	ct_eob_br															
	[T=3] [B=1] [R=1] [CB=3] [CT=0]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=3]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=4]	[T=3] [B=1] [R=1] [CB=5]	[T=3] [B=1] [R=1] [CB=5]	[T=3] [B=1] [R=1] [CB=5]	[T=3] [B=1] [R=1] [CB=5] [CT=3]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		[CT=1]	[CT=2]	[CT=3]	[CT=4]	[CT=5]	[CT=0]	[CT=1]	[CT=2]	[CT=3]	[CT=4]	[CT=5]	[CT=0]	[CT=1]	[CT=2]	
R185	ct_eob_br		ct_uv_mode													
	[T=3] [B=1] [R=1] [CB=5] [CT=4]	[T=3] [B=1] [R=1] [CB=5] [CT=5]	[0] [0]	[0] [1]	[0] [2]	[0] [3]	[0] [4]	[0] [5]	[0] [6]	[0] [7]	[0] [8]	[0] [9]	[1] [0]	[1] [1]	[1] [2]	[1] [3]
R186	ct_uv_mode															
	[1] [4]	[1] [5]	[1] [6]	[1] [7]	[1] [8]	[1] [9]	[2] [0]	[2] [1]	[2] [2]	[2] [3]	[2] [4]	[2] [5]	[2] [6]	[2] [7]	[2] [8]	[2] [9]
R187	ct_uv_mode															
	[3] [0]	[3] [1]	[3] [2]	[3] [3]	[3] [4]	[3] [5]	[3] [6]	[3] [7]	[3] [8]	[3] [9]	[4] [0]	[4] [1]	[4] [2]	[4] [3]	[4] [4]	[4] [5]
R188	ct_uv_mode															
	[4] [6]	[4] [7]	[4] [8]	[4] [9]	[5] [0]	[5] [1]	[5] [2]	[5] [3]	[5] [4]	[5] [5]	[5] [6]	[5] [7]	[5] [8]	[5] [9]	[6] [0]	[6] [1]
R189	ct_uv_mode															
	[6] [2]	[6] [3]	[6] [4]	[6] [5]	[6] [6]	[6] [7]	[6] [8]	[6] [9]	[7] [0]	[7] [1]	[7] [2]	[7] [3]	[7] [4]	[7] [5]	[7] [6]	[7] [7]
R190	ct_uv_mode															
	[7] [8]	[7] [9]	[8] [0]	[8] [1]	[8] [2]	[8] [3]	[8] [4]	[8] [5]	[8] [6]	[8] [7]	[8] [8]	[8] [9]	[9] [0]	[9] [1]	[9] [2]	[9] [3]
R191	ct_uv_mode						ct_seg_id								ct_seg_pred	
	[9]	[9]	[9]	[9]	[9]	[9]	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[0]	[0]

Rxxx Value 1 row = 1 Cacheline	16 32-bit counters															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	[4]	[5]	[6]	[7]	[8]	[9]									[0]	[1]
R192	ct_seg_pred															
	[1] [0]	[1] [1]	[2] [0]	[2] [1]												

Probability Statistics Counters Context Legend

Counter	i	j	k	l	m	n	Dimension	Total Number of Counters
ct_tx_8x8 [i] [j]	Context	Node					2X2	4
ct_tx_16x16 [i] [j]	Context	Node					2X3	6
ct_tx_32x32 [i] [j]	Context	Node					2X4	8
ct_coef [T=i] [B=j] [R=k] [CB=l] [CT=m] [N=n]	T = TransformSize (0 4x4, 1 8x8, 2 16x16, 3 32x32)	B = BlockType (0 Y, 1 UV)	R = ReferenceType (0 Intra, 1 Inter)	CB = CoefficientBand	CT = Context	N = Node (0 ZERO, 1 ONE, 2 TWO, 3	4X2X2X6X6X4	2112

Counter	i	j	k	l	m	n	Dimension	Total Number of Counters
						EOB)		
ct_mbskip [i] [j]	Context	Node					3X2	6
ct_inter_mode [i] [j]	Context (BOTH_ZERO, ZERO_PLUS_PREDICTED, BOTH_PREDICTED, NEW_PLUS_NON_INTRA, BOTH_NEW, INTRA_PLUS_NON_INTRA, BOTH_INTRA)	Node (NEARESTMV, NEARMV, ZEROMV, NEWMV)					7X4	28
ct_switchable_interp [i] [j]	Context	Node (MV_JOINT_ZERO, MV_JOINT_HNZVZ, MV_JOINT_HZVNZ, MV_JOINT_HNZVNZ)					3X4	12
ct_intra_inter [i] [j]	Context	Node					4X2	8
ct_comp_inter [i] [j]	Context	Node					5X2	10
ct_single_ref [i] [j] [k]	P0,P1	Context	Node				2X5X2	20
ct_comp_ref	Context	Node					5X2	10

Counter	i	j	k	l	m	n	Dimension	Total Number of Counters
[i] [j]								
ct_y_mode [i] [j]	BLOCK_SIZE_GROUPS	Node (DC_PRED, TM_PRED, V_PRED, H_PRED, D135_PRED, D117_PRED, D45_PRED, D63_PRED, D153_PRED, D27_PRED)					4x10	40
ct_partition [i] [j]	Context	Node (00: NONE, 01: HORZ, 10: VERT, 11: SPLIT)					16X4	64
ct_mv_joints [i]	Node (MV_JOINT_ZERO, MV_JOINT_HNZVZ, MV_JOINT_HZVNZ, MV_JOINT_HNZVNZ)						4	4
ct_mv_sign [i] [j]	XY	Node					2X2	4
ct_mv_classes [i] [j]	XY	MV CLASS					2X10	20
ct_mv_class0 [i] [j]	XY	Node					2X2	4
ct_mv_bits	XY	MV BIT POSITION	Node				2X10X2	40

Counter	i	j	k	l	m	n	Dimension	Total Number of Counters
[i] [j] [k]								
ct_mv_class0_fp	XY	MV_CLASS0 BIT	Node				2X2X4	16
[i] [j] [k]								
ct_mv_fp	XY	Node					2X4	8
[i] [j]								
ct_mv_class0_hp	XY	Node					2X2	4
[i] [j]								
ct_mv_hp	XY	Node					2X2	4
[i] [j]								
ct_eob_br	T = TransformSize (0 4x4, 1 8x8, 2 16x16, 3 32x32)	B = BlockType (0 Y, 1 UV)	R = ReferenceType (0 Intra, 1 Inter)	CB = CoefficientBand	CT = Context		4X2X2X6X6	528
[T=i] [B=j] [R=k] [CB=l] [CT=m]								
ct_uv_mode	Context	Node(DC_PRED, TM_PRED, V_PRED, H_PRED, D135_PRED, D117_PRED, D45_PRED,					10x10	100
[i] [j]								

Counter	i	j	k	l	m	n	Dimension	Total Number of Counters
		D63_PRED, D153_PRED, D27_PRED)						
ct_seg_id[i] [i]	Node						8	8
ct_seg_pred [i] [j]	Context	Node					3X2	6
								3074

VP9 PAK Quant Lookup Tables

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
0	4	4	4	4	4	4	1	1	1	1	1	1	32768	32768	32768	32768	32768	32768
1	8	8	9	9	12	13	2	2	2	2	2	2	32768	32768	29127	29127	21845	20164
2	8	9	10	11	18	19	2	2	2	2	3	3	32768	29127	26214	23831	29127	27594
3	9	10	13	13	25	27	2	2	2	2	3	3	29127	26214	20164	20164	20971	19418
4	10	11	15	16	33	35	2	2	2	3	4	4	26214	23831	17476	32768	31775	29959
5	11	12	17	18	41	44	2	2	3	3	4	4	23831	21845	30840	29127	25575	23831
6	12	13	20	21	50	54	2	2	3	3	4	4	21845	20164	26214	24966	20971	19418
7	12	14	22	24	60	64	2	2	3	3	4	5	21845	18724	23831	21845	17476	32768
8	13	15	25	27	70	75	2	2	3	3	5	5	20164	17476	20971	19418	29959	27962
9	14	16	28	30	80	87	2	3	3	3	5	5	18724	32768	18724	17476	26214	24105
10	15	17	31	33	91	99	2	3	3	4	5	5	17476	30840	16912	31775	23045	21183
11	16	18	34	37	103	112	3	3	4	4	5	5	32768	29127	30840	28339	20360	18724
12	17	19	37	40	115	126	3	3	4	4	5	5	30840	27594	28339	26214	18236	16644
13	18	20	40	44	127	139	3	3	4	4	5	6	29127	26214	26214	23831	16513	30174
14	19	21	43	48	140	154	3	3	4	4	6	6	27594	24966	24385	21845	29959	27235
15	19	22	47	51	153	168	3	3	4	4	6	6	27594	23831	22310	20560	27413	24966
16	20	23	50	55	166	183	3	3	4	4	6	6	26214	22795	20971	19065	25266	22919
17	21	24	53	59	180	199	3	3	4	4	6	6	24966	21845	19784	17772	23301	21076

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
18	22	25	57	63	194	214	3	3	4	4	6	6	23831	20971	18396	16644	21620	19599
19	23	26	60	67	208	230	3	3	4	5	6	6	22795	20164	17476	31300	20164	18236
20	24	27	64	71	222	247	3	3	5	5	6	6	21845	19418	32768	29537	18893	16980
21	25	28	68	75	237	263	3	3	5	5	6	7	20971	18724	30840	27962	17697	31895
22	26	29	71	79	251	280	3	3	5	5	6	7	20164	18078	29537	26546	16710	29959
23	26	30	75	83	266	297	3	3	5	5	7	7	20164	17476	27962	25266	31536	28244
24	27	31	78	88	281	314	3	3	5	5	7	7	19418	16912	26886	23831	29852	26715
25	28	32	82	92	296	331	3	4	5	5	7	7	18724	32768	25575	22795	28339	25343
26	29	33	86	96	312	349	3	4	5	5	7	7	18078	31775	24385	21845	26886	24036
27	30	34	90	100	327	366	3	4	5	5	7	7	17476	30840	23301	20971	25653	22919
28	31	35	93	105	343	384	3	4	5	5	7	7	16912	29959	22550	19972	24456	21845
29	32	36	97	109	358	402	4	4	5	5	7	7	32768	29127	21620	19239	23431	20867
30	32	37	101	114	374	420	4	4	5	5	7	7	32768	28339	20763	18396	22429	19972
31	33	38	105	118	390	438	4	4	5	5	7	7	31775	27594	19972	17772	21509	19152
32	34	39	109	122	405	456	4	4	5	5	7	7	30840	26886	19239	17189	20712	18396
33	35	40	113	127	421	475	4	4	5	5	7	7	29959	26214	18558	16513	19925	17660
34	36	41	116	131	437	493	4	4	5	6	7	7	29127	25575	18078	32017	19195	17015
35	37	42	120	136	453	511	4	4	5	6	7	7	28339	24966	17476	30840	18517	16416
36	38	43	124	140	469	530	4	4	5	6	7	8	2759	2438	1691	2995	1788	3165

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
													4	5	2	9	6	5
37	38	44	128	145	484	548	4	4	6	6	7	8	27594	23831	32768	28926	17331	30615
38	39	45	132	149	500	567	4	4	6	6	7	8	26886	23301	31775	28149	16777	29589
39	40	46	136	154	516	586	4	4	6	6	8	8	26214	22795	30840	27235	32513	28630
40	41	47	140	158	532	604	4	4	6	6	8	8	25575	22310	29959	26546	31536	27776
41	42	48	143	163	548	623	4	4	6	6	8	8	24966	21845	29330	25731	30615	26929
42	43	49	147	168	564	642	4	4	6	6	8	8	24385	21399	28532	24966	29746	26132
43	43	50	151	172	580	660	4	4	6	6	8	8	24385	20971	27776	24385	28926	25420
44	44	51	155	177	596	679	4	4	6	6	8	8	23831	20560	27060	23696	28149	24708
45	45	52	159	181	611	698	4	4	6	6	8	8	23301	20164	26379	23172	27458	24036
46	46	53	163	186	627	716	4	4	6	6	8	8	22795	19784	25731	22550	26757	23431
47	47	54	166	190	643	735	4	4	6	6	8	8	22310	19418	25266	22075	26092	22826
48	48	55	170	195	659	753	4	4	6	6	8	8	21845	19065	24672	21509	25458	22280
49	48	56	174	199	674	772	4	4	6	6	8	8	21845	18724	24105	21076	24892	21732
50	49	57	178	204	690	791	4	4	6	6	8	8	21399	18396	23563	20560	24314	21210
51	50	58	182	208	706	809	4	4	6	6	8	8	20971	18078	23045	20164	23763	20738
52	51	59	185	213	721	828	4	4	6	6	8	8	20560	17772	22671	19691	23269	20262
53	52	60	189	217	737	846	4	4	6	6	8	8	20164	17476	22192	19328	22764	19831
54	53	61	193	222	752	865	4	4	6	6	8	8	19784	17189	21732	18893	22310	19395

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
55	53	62	197	226	768	884	4	4	6	6	8	8	19784	16912	21290	18558	21845	18978
56	54	63	200	231	783	902	4	4	6	6	8	8	19418	16644	20971	18157	21426	18600
57	55	64	204	235	798	920	4	5	6	6	8	8	19065	32768	20560	17848	21024	18236
58	56	65	208	240	814	939	4	5	6	6	8	8	18724	32263	20164	17476	20610	17867
59	57	66	212	244	829	957	4	5	6	6	8	8	18396	31775	19784	17189	20237	17531
60	57	67	215	249	844	976	4	5	6	6	8	8	18396	31300	19508	16844	19878	17189
61	58	68	219	253	859	994	4	5	6	6	8	8	18078	30840	19152	16578	19531	16878
62	59	69	223	258	874	1012	4	5	6	7	8	8	17772	30393	18808	32513	19195	16578
63	60	70	226	262	889	1030	4	5	6	7	8	9	17476	29959	18558	32017	18872	32577
64	61	71	230	267	904	1049	4	5	6	7	8	9	17189	29537	18236	31418	18558	31987
65	62	72	233	271	919	1067	4	5	6	7	8	9	16912	29127	18001	30954	18255	31447
66	62	73	237	275	934	1085	4	5	6	7	8	9	16912	28728	17697	30504	17962	30925
67	63	74	241	280	949	1103	4	5	6	7	8	9	16644	28339	17403	29959	17678	30421
68	64	75	244	284	964	1121	5	5	6	7	8	9	32768	27962	17189	29537	17403	29932
69	65	76	248	289	978	1139	5	5	6	7	8	9	32263	27594	16912	29026	17154	29459
70	66	77	251	293	993	1157	5	5	6	7	8	9	31775	27235	16710	28630	16895	29001
71	66	78	255	297	1008	1175	5	5	6	7	8	9	31775	26886	16448	28244	16644	28556
72	67	79	259	302	1022	1193	5	5	7	7	8	9	31300	26546	32388	27776	16416	28126
73	68	80	262	306	1037	1211	5	5	7	7	9	9	3084	2621	3201	2741	3235	2770

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
													0	4	7	3	7	8
74	69	81	266	311	1051	1229	5	5	7	7	9	9	30393	25890	31536	26973	31926	27302
75	70	82	269	315	1065	1246	5	5	7	7	9	9	29959	25575	31184	26630	31506	26929
76	70	83	273	319	1080	1264	5	5	7	7	9	9	29959	25266	30727	26296	31068	26546
77	71	84	276	324	1094	1282	5	5	7	7	9	9	29537	24966	30393	25890	30671	26173
78	72	85	280	328	1108	1299	5	5	7	7	9	9	29127	24672	29959	25575	30283	25830
79	73	86	283	332	1122	1317	5	5	7	7	9	9	28728	24385	29641	25266	29905	25477
80	74	87	287	337	1136	1335	5	5	7	7	9	9	28339	24105	29228	24892	29537	25134
81	74	88	290	341	1151	1352	5	5	7	7	9	9	28339	23831	28926	24600	29152	24818
82	75	89	293	345	1165	1370	5	5	7	7	9	9	27962	23563	28630	24314	28802	24492
83	76	90	297	349	1179	1387	5	5	7	7	9	9	27594	23301	28244	24036	28460	24192
84	77	91	300	354	1192	1405	5	5	7	7	9	9	27235	23045	27962	23696	28149	23882
85	78	92	304	358	1206	1422	5	5	7	7	9	9	26886	22795	27594	23431	27822	23596
86	78	93	307	362	1220	1440	5	5	7	7	9	9	26886	22550	27324	23172	27503	23301
87	79	94	310	367	1234	1457	5	5	7	7	9	9	26546	22310	27060	22857	27191	23029
88	80	95	314	371	1248	1474	5	5	7	7	9	9	26214	22075	26715	22610	26886	22764
89	81	96	317	375	1261	1491	5	5	7	7	9	9	25890	21845	26462	22369	26609	22504
90	81	97	321	379	1275	1509	5	5	7	7	9	9	25890	21620	26132	22133	26317	22236
91	82	98	324	384	1288	1526	5	5	7	7	9	9	25575	21399	25890	21845	26051	21988

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
92	83	99	327	388	1302	1543	5	5	7	7	9	9	25266	21183	25653	21620	25771	21746
93	84	100	331	392	1315	1560	5	5	7	7	9	9	24966	20971	25343	21399	25516	21509
94	85	101	334	396	1329	1577	5	5	7	7	9	9	24672	20763	25115	21183	25247	21277
95	85	102	337	401	1342	1595	5	5	7	7	9	9	24672	20560	24892	20919	25003	21037
96	87	104	343	409	1368	1627	5	5	7	7	9	9	24105	20164	24456	20510	24528	20623
97	88	106	350	417	1393	1660	5	5	7	7	9	9	23831	19784	23967	20116	24087	20213
98	90	108	356	425	1419	1693	5	5	7	7	9	9	23301	19418	23563	19737	23646	19819
99	92	110	362	433	1444	1725	5	5	7	7	9	9	22795	19065	23172	19373	23237	19451
100	93	112	369	441	1469	1758	5	5	7	7	9	9	22550	18724	22733	19021	22841	19086
101	95	114	375	449	1494	1791	5	5	7	7	9	9	22075	18396	22369	18682	22459	18735
102	96	116	381	458	1519	1824	5	5	7	7	9	9	21845	18078	22017	18315	22089	18396
103	98	118	387	466	1544	1856	5	5	7	7	9	9	21399	17772	21675	18001	21732	18078
104	99	120	394	474	1569	1889	5	5	7	7	9	9	21183	17476	21290	17697	21385	17763
105	101	122	400	482	1594	1922	5	5	7	7	9	9	20763	17189	20971	17403	21050	17458
106	102	124	406	490	1618	1954	5	5	7	7	9	9	20560	16912	20661	17119	20738	17172
107	104	126	412	498	1643	1987	5	5	7	7	9	9	20164	16644	20360	16844	20422	16886
108	105	128	418	506	1668	2020	5	6	7	7	9	9	19972	32768	20068	16578	20116	16611
109	107	130	424	514	1692	2052	5	6	7	8	9	10	19599	32263	19784	32640	19831	32704
110	108	132	430	523	1717	2085	5	6	7	8	9	10	1941	3177	1950	3207	1954	3218

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
													8	5	8	8	2	6
111	110	134	436	531	1741	2118	5	6	7	8	9	10	19065	31300	19239	31595	19273	31685
112	111	136	442	539	1765	2150	5	6	7	8	9	10	18893	30840	18978	31126	19011	31213
113	113	138	448	547	1789	2183	5	6	7	8	9	10	18558	30393	18724	30671	18755	30741
114	114	140	454	555	1814	2216	5	6	7	8	9	10	18396	29959	18477	30229	18497	30283
115	116	142	460	563	1838	2248	5	6	7	8	9	10	18078	29537	18236	29799	18255	29852
116	117	144	466	571	1862	2281	5	6	7	8	9	10	17924	29127	18001	29382	18020	29420
117	118	146	472	579	1885	2313	5	6	7	8	9	10	17772	28728	17772	28976	17800	29013
118	120	148	478	588	1909	2346	5	6	7	8	9	10	17476	28339	17549	28532	17576	28605
119	121	150	484	596	1933	2378	5	6	7	8	9	10	17331	27962	17331	28149	17358	28220
120	123	152	490	604	1957	2411	5	6	7	8	9	10	17050	27594	17119	27776	17145	27834
121	125	155	499	616	1992	2459	5	6	7	8	9	10	16777	27060	16810	27235	16844	27291
122	127	158	507	628	2027	2508	5	6	7	8	9	10	16513	26546	16545	26715	16553	26757
123	129	161	516	640	2061	2556	6	6	8	8	10	10	32513	26051	32513	26214	32561	26255
124	131	164	525	652	2096	2605	6	6	8	8	10	10	32017	25575	31956	25731	32017	25761
125	134	167	533	664	2130	2653	6	6	8	8	10	10	31300	25115	31476	25266	31506	25295
126	136	170	542	676	2165	2701	6	6	8	8	10	10	30840	24672	30954	24818	30997	24845
127	138	173	550	688	2199	2750	6	6	8	8	10	10	30393	24244	30504	24385	30517	24403
128	140	176	559	700	2233	2798	6	6	8	8	10	10	29959	23831	30012	23967	30053	23984

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
129	142	179	567	713	2267	2847	6	6	8	8	10	10	29537	23431	29589	23530	29602	23571
130	144	182	576	725	2300	2895	6	6	8	8	10	10	29127	23045	29127	23140	29177	23180
131	146	185	584	737	2334	2943	6	6	8	8	10	10	28728	22671	28728	22764	28752	22802
132	148	188	592	749	2367	2992	6	6	8	8	10	10	28339	22310	28339	22399	28351	22429
133	150	191	601	761	2400	3040	6	6	8	8	10	10	27962	21959	27915	22046	27962	22075
134	152	194	609	773	2434	3088	6	6	8	8	10	10	27594	21620	27548	21704	27571	21732
135	154	197	617	785	2467	3137	6	6	8	8	10	10	27235	21290	27191	21372	27202	21392
136	156	200	625	797	2499	3185	6	6	8	8	10	10	26886	20971	26843	21050	26854	21070
137	158	203	634	809	2532	3234	6	6	8	8	10	10	26546	20661	26462	20738	26504	20751
138	161	207	644	825	2575	3298	6	6	8	8	10	10	26051	20262	26051	20336	26061	20348
139	164	211	655	841	2618	3362	6	6	8	8	10	10	25575	19878	25614	19949	25633	19960
140	166	215	666	857	2661	3426	6	6	8	8	10	10	25266	19508	25191	19576	25219	19588
141	169	219	676	873	2704	3491	6	6	8	8	10	10	24818	19152	24818	19217	24818	19223
142	172	223	687	889	2746	3555	6	6	8	8	10	10	24385	18808	24420	18872	24438	18877
143	174	227	698	905	2788	3619	6	6	8	8	10	10	24105	18477	24036	18538	24070	18543
144	177	231	708	922	2830	3684	6	6	8	8	10	10	23696	18157	23696	18196	23713	18216
145	180	235	718	938	2872	3748	6	6	8	8	10	10	23301	17848	23366	17886	23366	17905
146	182	239	729	954	2913	3812	6	6	8	8	10	10	23045	17549	23014	17586	23037	17604
147	185	243	739	970	2954	3876	6	6	8	8	10	10	2267	1726	2270	1729	2271	1731

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
													1	0	2	6	7	3
148	187	247	749	986	2995	3941	6	6	8	8	10	10	22429	16980	22399	17015	22406	17028
149	190	251	759	1002	3036	4005	6	6	8	8	10	10	22075	16710	22104	16743	22104	16756
150	192	255	770	1018	3076	4069	6	6	8	8	10	10	21845	16448	21788	16480	21816	16492
151	195	260	782	1038	3127	4149	6	7	8	9	10	11	21509	32263	21454	32326	21461	32349
152	199	265	795	1058	3177	4230	6	7	8	9	10	11	21076	31655	21103	31714	21123	31729
153	202	270	807	1078	3226	4310	6	7	8	9	10	11	20763	31068	20789	31126	20802	31141
154	205	275	819	1098	3275	4390	6	7	8	9	10	11	20460	30504	20485	30559	20491	30573
155	208	280	831	1118	3324	4470	6	7	8	9	10	11	20164	29959	20189	30012	20189	30026
156	211	285	844	1138	3373	4550	6	7	8	9	10	11	19878	29433	19878	29485	19895	29498
157	214	290	856	1158	3421	4631	6	7	8	9	10	11	19599	28926	19599	28976	19616	28982
158	217	295	868	1178	3469	4711	6	7	8	9	10	11	19328	28435	19328	28484	19345	28490
159	220	300	880	1198	3517	4791	6	7	8	9	10	11	19065	27962	19065	28008	19081	28014
160	223	305	891	1218	3565	4871	6	7	8	9	10	11	18808	27503	18829	27548	18824	27554
161	226	311	906	1242	3621	4967	6	7	8	9	10	11	18558	26973	18517	27016	18533	27021
162	230	317	920	1266	3677	5064	6	7	8	9	10	11	18236	26462	18236	26504	18250	26504
163	233	323	933	1290	3733	5160	6	7	8	9	10	11	18001	25970	17982	26011	17977	26011
164	237	329	947	1314	3788	5256	6	7	8	9	10	11	17697	25497	17716	25536	17716	25536
165	240	335	961	1338	3843	5352	6	7	8	9	10	11	17476	25040	17458	25078	17462	25078

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
166	243	341	975	1362	3897	5448	6	7	8	9	10	11	17260	24600	17207	24636	17220	24636
167	247	347	988	1386	3951	5544	6	7	8	9	10	11	16980	24174	16980	24209	16985	24209
168	250	353	1001	1411	4005	5641	6	7	8	9	10	11	16777	23763	16760	23780	16756	23793
169	253	359	1015	1435	4058	5737	6	7	8	9	10	11	16578	23366	16529	23382	16537	23395
170	257	366	1030	1463	4119	5849	7	7	9	9	11	11	32640	22919	32577	22935	32585	22947
171	261	373	1045	1491	4181	5961	7	7	9	9	11	11	32140	22489	32109	22504	32101	22515
172	265	380	1061	1519	4241	6073	7	7	9	9	11	11	31655	22075	31625	22089	31647	22100
173	269	387	1076	1547	4301	6185	7	7	9	9	11	11	31184	21675	31184	21690	31206	21700
174	272	394	1090	1575	4361	6297	7	7	9	9	11	11	30840	21290	30783	21304	30776	21314
175	276	401	1105	1603	4420	6410	7	7	9	9	11	11	30393	20919	30366	20932	30366	20938
176	280	408	1120	1631	4479	6522	7	7	9	9	11	11	29959	20560	29959	20572	29966	20579
177	284	416	1137	1663	4546	6650	7	7	9	9	11	11	29537	20164	29511	20177	29524	20183
178	288	424	1153	1695	4612	6778	7	7	9	9	11	11	29127	19784	29101	19796	29101	19801
179	292	432	1170	1727	4677	6906	7	7	9	9	11	11	28728	19418	28679	19429	28697	19434
180	296	440	1186	1759	4742	7034	7	7	9	9	11	11	28339	19065	28292	19075	28304	19081
181	300	448	1202	1791	4807	7162	7	7	9	9	11	11	27962	18724	27915	18735	27921	18740
182	304	456	1218	1823	4871	7290	7	7	9	9	11	11	27594	18396	27548	18406	27554	18411
183	309	465	1236	1859	4942	7435	7	7	9	9	11	11	27147	18040	27147	18049	27158	18052
184	313	474	125	189	5013	7579	7	7	9	9	11	11	2680	1769	2677	1770	2677	1770

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
			3	5									0	7	9	6	3	9
185	317	483	1271	1931	5083	7723	7	7	9	9	11	11	26462	17367	26400	17376	26405	17378
186	322	492	1288	1967	5153	7867	7	7	9	9	11	11	26051	17050	26051	17058	26046	17060
187	326	501	1306	2003	5222	8011	7	7	9	9	11	11	25731	16743	25692	16752	25702	16754
188	330	510	1323	2039	5291	8155	7	7	9	9	11	11	25420	16448	25362	16456	25367	16458
189	335	520	1342	2079	5367	8315	7	8	9	10	11	12	25040	32263	25003	32279	25007	32283
190	340	530	1361	2119	5442	8475	7	8	9	10	11	12	24672	31655	24654	31670	24663	31673
191	344	540	1379	2159	5517	8635	7	8	9	10	11	12	24385	31068	24332	31083	24328	31086
192	349	550	1398	2199	5591	8795	7	8	9	10	11	12	24036	30504	24001	30517	24006	30521
193	354	560	1416	2239	5665	8956	7	8	9	10	11	12	23696	29959	23696	29972	23692	29972
194	359	571	1436	2283	5745	9132	7	8	9	10	11	12	23366	29382	23366	29395	23362	29395
195	364	582	1456	2327	5825	9308	7	8	9	10	11	12	23045	28826	23045	28839	23041	28839
196	369	593	1476	2371	5905	9484	7	8	9	10	11	12	22733	28292	22733	28304	22729	28304
197	374	604	1496	2415	5984	9660	7	8	9	10	11	12	22429	27776	22429	27788	22429	27788
198	379	615	1516	2459	6063	9836	7	8	9	10	11	12	22133	27280	22133	27291	22137	27291
199	384	627	1537	2507	6149	10028	7	8	9	10	11	12	21845	26757	21831	26768	21827	26768
200	389	639	1559	2555	6234	10220	7	8	9	10	11	12	21564	26255	21523	26265	21529	26265
201	395	651	1580	2603	6319	10412	7	8	9	10	11	12	21236	25771	21236	25781	21240	25781
202	400	663	1601	2651	6404	10604	7	8	9	10	11	12	20971	25305	20958	25314	20958	25314

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
203	406	676	1624	2703	6495	10812	7	8	9	10	11	12	20661	24818	20661	24827	20664	24827
204	411	689	1647	2755	6587	11020	7	8	9	10	11	12	20410	24350	20373	24358	20376	24358
205	417	702	1670	2807	6678	11228	7	8	9	10	11	12	20116	23899	20092	23907	20098	23907
206	423	715	1692	2859	6769	11437	7	8	9	10	11	12	19831	23464	19831	23472	19828	23470
207	429	729	1717	2915	6867	11661	7	8	9	10	11	12	19553	23014	19542	23021	19545	23019
208	435	743	1741	2971	6966	11885	7	8	9	10	11	12	19284	22580	19273	22587	19267	22586
209	441	757	1766	3027	7064	12109	7	8	9	10	11	12	19021	22162	19000	22170	19000	22168
210	447	771	1791	3083	7163	12333	7	8	9	10	11	12	18766	21760	18735	21767	18737	21765
211	454	786	1817	3143	7269	12573	7	8	9	10	11	12	18477	21345	18466	21351	18464	21350
212	461	801	1844	3203	7376	12813	7	8	9	10	11	12	18196	20945	18196	20951	18196	20950
213	467	816	1871	3263	7483	13053	7	8	9	10	11	12	17962	20560	17933	20566	17936	20565
214	475	832	1900	3327	7599	13309	7	8	9	10	11	12	17660	20164	17660	20170	17662	20169
215	482	848	1929	3391	7715	13565	7	8	9	10	11	12	17403	19784	17394	19790	17396	19788
216	489	864	1958	3455	7832	13821	7	8	9	10	11	12	17154	19418	17137	19423	17137	19422
217	497	881	1990	3523	7958	14093	7	8	9	10	11	12	16878	19043	16861	19048	16865	19047
218	505	898	2021	3591	8085	14365	7	8	9	10	11	12	16611	18682	16602	18688	16600	18686
219	513	915	2054	3659	8214	14637	8	8	10	10	12	12	32704	18335	32672	18340	32680	18339
220	522	933	2088	3731	8352	14925	8	8	10	10	12	12	32140	17982	32140	17986	32140	17985
221	530	951	212	380	8492	1521	8	8	10	10	12	12	3165	1764	3161	1764	3161	1764

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
			3	3		3							5	1	0	6	0	5
222	539	969	2159	3876	8635	15502	8	8	10	10	12	12	31126	17313	31083	17313	31086	17316
223	549	988	2197	3952	8788	15806	8	8	10	10	12	12	30559	16980	30545	16980	30545	16983
224	559	1007	2236	4028	8945	16110	8	8	10	10	12	12	30012	16660	30012	16660	30009	16662
225	569	1026	2276	4104	9104	16414	8	9	10	11	12	13	29485	32704	29485	32704	29485	32708
226	579	1046	2319	4184	9275	16734	8	9	10	11	12	13	28976	32078	28938	32078	28941	32082
227	590	1066	2363	4264	9450	17054	8	9	10	11	12	13	28435	31476	28399	31476	28405	31480
228	602	1087	2410	4348	9639	17390	8	9	10	11	12	13	27869	30868	27846	30868	27848	30872
229	614	1108	2458	4432	9832	17726	8	9	10	11	12	13	27324	30283	27302	30283	27302	30287
230	626	1129	2508	4516	10031	18062	8	9	10	11	12	13	26800	29720	26757	29720	26760	29723
231	640	1151	2561	4604	10245	18414	8	9	10	11	12	13	26214	29152	26204	29152	26201	29155
232	654	1173	2616	4692	10465	18766	8	9	10	11	12	13	25653	28605	25653	28605	25650	28608
233	668	1196	2675	4784	10702	19134	8	9	10	11	12	13	25115	28055	25087	28055	25082	28058
234	684	1219	2737	4876	10946	19502	8	9	10	11	12	13	24528	27526	24519	27526	24523	27529
235	700	1243	2802	4972	11210	19886	8	9	10	11	12	13	23967	26994	23950	26994	23946	26997
236	717	1267	2871	5068	11482	20270	8	9	10	11	12	13	23399	26483	23374	26483	23378	26485
237	736	1292	2944	5168	11776	20670	8	9	10	11	12	13	22795	25970	22795	25970	22795	25973
238	755	1317	3020	5268	12081	21070	8	9	10	11	12	13	22221	25477	22221	25477	22219	25480
239	775	1343	3102	5372	12409	21486	8	9	10	11	12	13	21648	24984	21634	24984	21632	24987

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
240	796	1369	3188	5476	12750	21902	8	9	10	11	12	13	21076	24510	21050	24510	21053	24512
241	819	1396	3280	5584	13118	22334	8	9	10	11	12	13	20485	24036	20460	24036	20463	24038
242	843	1423	3375	5692	13501	22766	8	9	10	11	12	13	19901	23580	19884	23580	19882	23582
243	869	1451	3478	5804	13913	23214	8	9	10	11	12	13	19306	23125	19295	23125	19293	23127
244	896	1479	3586	5916	14343	23662	8	9	10	11	12	13	18724	22687	18714	22687	18715	22689
245	925	1508	3702	6032	14807	24126	8	9	10	11	12	13	18137	22250	18127	22250	18128	22252
246	955	1537	3823	6148	15290	24590	8	9	10	11	12	13	17567	21831	17553	21831	17556	21832
247	988	1567	3953	6268	15812	25070	8	9	10	11	12	13	16980	21413	16976	21413	16976	21414
248	1022	1597	4089	6388	16356	25551	8	9	10	11	12	13	16416	21010	16412	21010	16412	21011
249	1058	1628	4236	6512	16943	26047	9	9	11	11	13	13	31714	20610	31685	20610	31686	20611
250	1098	1660	4394	6640	17575	26559	9	9	11	11	13	13	30559	20213	30545	20213	30547	20214
251	1139	1692	4559	6768	18237	27071	9	9	11	11	13	13	29459	19831	29440	19831	29438	19831
252	1184	1725	4737	6900	18949	27599	9	9	11	11	13	13	28339	19451	28333	19451	28332	19452
253	1232	1759	4929	7036	19718	28143	9	9	11	11	13	13	27235	19075	27230	19075	27227	19076
254	1282	1793	5130	7172	20521	28687	9	9	11	11	13	13	26173	18714	26163	18714	26162	18714
255	1336	1828	5347	7312	21387	29247	9	9	11	11	13	13	25115	18355	25101	18355	25102	18356

SB, CU/PU and TU Sizes – Encoder Only

CU/PU/TU Partitioning Configurations

SB size	CU size	min/max TU range
64x64	64x64	32x32..4x4
	32x32	32x32..4x4
	16x16	16x16..4x4
	8x8	8x8..4x4

PU Options for a Given CU

Current CU size	Possible CU sizes	Allowed CU/PU partition types.
64x64	64x64	2Nx2N, 2NxN, Nx2N
	32x32	2Nx2N, 2NxN, Nx2N
	16x16	2Nx2N, 2NxN, Nx2N
	8x8	2Nx2N, 2NxN, Nx2N, NxN

Definition of the VP9 CU Record Structure - Encoder Only

The following table defines the CU record structure as indirect data to the PAK Object Command. Entries are DW based (4 bytes) and cache aligned. This memory surface is pointed to by the HCP Indirect CU Object Base Address in the HCP_IND_OBJ_BASE_ADDR_STATE Command.

VP9 CU Record Structure Definition (UnPacked CU)

DW	Bit	Field	Bits	Definition	Comments
0	1:0	cu_size	2	0: 8x8, 1:16x16, 2: 32x32, and 3: 64x64	Indicates size of the CU
	3:2	Reserved	4	MBZ	
	5:4	cu_part_mode	2	0: 2Nx2N, 1: 2NxN, 2: Nx2N, 3: NxN (8x8 only)	
	7:6	Reserved	2	MBZ	
	11:8	intra_chroma_mode[0][3:0]	4	0:DC_PRED 1:V_PRED 2:H_PRED 3:TM_PRED 4:D45_PRED 5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED 9:D63_PRED 10-15 Reserved	
	15:12	Reserved	4	MBZ	
	19:16	intra_chroma_mode[1][3:0]	4	0:DC_PRED 1:V_PRED 2:H_PRED 3:TM_PRED 4:D45_PRED	Applicable for part1 of shapes > 8x8

DW	Bit	Field	Bits	Definition	Comments
				5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED 9:D63_PRED 10-15 Reserved	
	20	cu_pred_mode0	1	0: Intra 1: Inter	Pred mode for part0 in raster scan order cu_pred_mode=intra means ref_reframe0=intra
	21	cu_pred_mode1	1	0: Intra 1: Inter	Pred mode for part1 in raster scan order cu_pred_mode=intra means ref_reframe0=intra
	23:22	Reserved	2	MBZ	
	24	interpred_comp0	1	0: Single 1: Compound(ref_reframe1>intra)	Interpred comp mode for Part0
	25	interpred_comp1	0	0: Single 1: Compound(ref_reframe1>intra)	Interpred comp mode for Part1
	31:26	Reserved	6	MBZ	
1	3:0	intra_mode[0][3:0]	4	0:DC_PRED 1:V_PRED 2:H_PRED 3:TM_PRED 4:D45_PRED 5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED 9:D63_PRED	Luma Intra mode for part0
	7:4	Reserved	4		
	11:8	intra_mode[1][3:0]	4		Luma pred mode for part1
	15:12	Reserved	4		
	19:16	intra_mode[2][3:0]	4		Luma pred mode for part2;

DW	Bit	Field	Bits	Definition	Comments
					applicable for 4x4 only
	23:19	Reserved	4		
	27:24	intra_mode[3][3:0]	4		Luma pred mode for part3; applicable for 4x4 only
	31:28	Reserved	4		
2	15:0	mvx_reframe0[0][15:0]	16		Frame0(Forward) MVx for part0 16-bit each(2's complement with 3 fractional bits); encodable range is [-16382,16382]).
	31:16	mvy_reframe0[0][15:0]	16		Frame0(Forward) MVy for part0 16-bit each(2's complement with 3 fractional bits); encodable range is [-16382,16382]).
3	15:0	mvx_reframe0[1][15:0]	16		Frame0 (Forward) MVx for part1 16-bit each(2's complement with 3 fractional bits); encodable range is [-16382,16382]).
	31:16	mvy_reframe0[1][15:0]	16		Frame0 (Forward) MVy for part1 16-bit each(2's complement with 3 fractional bits); encodable range is [-16382,16382]).
4	15:0	mvx_reframe0[2][15:0]	16		
	31:16	mvy_reframe0[2][15:0]	16		
5	15:0	mvx_reframe0[3][15:0]	16		
	31:16	mvy_reframe0[3][15:0]	16		
6	15:0	mvx_reframe1[0][15:0]	16		
	31:16	mvy_reframe1[0][15:0]	16		
7	15:0	mvx_reframe1[1][15:0]	16		
	31:16	mvy_reframe1[1][15:0]	16		
8	15:0	mvx_reframe1[2][15:0]	16		Interleaved X and Y
	31:16	mvy_reframe1[2][15:0]	16		
9	15:0	mvx_reframe1[3][15:0]	16		
	31:16	mvy_reframe1[3][15:0]	16		
10	1:0	ref_reframe0[0][1:0]	2	0:intra	frame0(forward)reference frame

DW	Bit	Field	Bits	Definition	Comments
				1:last 2:golden 3:altref	id for part0 HW uses if SegmentReferenceEnabled=0 in segment ID command
	3:2	Reserved	2	MBZ	
	5:4	ref_reframe0[1][1:0]	2	0:intra 1:last 2:golden 3:altref	frame0(forward)reference frame id for part1
	7:6	Reserved	2	MBZ	
	9:8	ref_reframe1 [0][1:0]	2	0:intra 1:last 2:golden 3:altref	frame1(backward)reference frame id for part0
	11:10	Reserved	2	MBZ	
	13:12	ref_reframe1 [1][1:0]	2	0:intra 1:last 2:golden 3:altref	frame0(forward)reference frame id for part1
	15:14	Reserved	2	MBZ	
	18:16	QuantRound0	3	0:+1/16 1:+2/16 2:+3/16 3:+4/16 4:+5/16 5:+6/16(default) 6:+7/16 7:+8/16	Quantization Round value for Part0 (8x8 and below shapes will have the same round value)
	19	Reserved		MBZ	
	22:20	QuantRound1	3	0:+1/16	Quantization Round value for Part1

DW	Bit	Field	Bits	Definition	Comments
				1:+2/16 2:+3/16 3:+4/16 4:+5/16 5:+6/16(default) 6:+7/16 7:+8/16	
	31:23	Reserved	1	MBZ	
11	1:0	tu_size[0][1:0]	2	0: 4x4 1: 8x8 2: 16x16 3: 32x32	tu_size[2][1:0], tu_size[3][1:0] must be 0 (4x4)
11	3:2	tu_size[1][1:0]	2	0: 4x4 1: 8x8 2: 16x16 3: 32x32	
11	13:4	Reserved	10	MBZ	
11	14	segmentPredFlag0	1	0:Diaable 1:Enable	Segment Prediction disable for Part0
11	15	segmentPredFlag1	1	0:Diaable 1:Enable	Segment Prediction disable for Part1
11	18:16	SegmentIdx[0][2:0]	3	Segment 0 to 7	SegmentID for part0 of blocks > 8x8
11	21:19	SegmentIdx[1][2:0]	3	Segment 0 to 7	SegmentID for part1 of blocks > 8x8
11	23:22	MotionComp_flttype[0][1:0]	2	0: EIGHTTAP 1: EIGHTTAP_SMOOTH 2: EIGHTTAP_SHARP 3: BILINEAR	HW will use this filtertype if SWITCHABLE=1 in pic state; Used for part0
11	25:24	MotionComp_flttype[1][1:0]	2	0: EIGHTTAP	For part1 of blocks >8x8

DW	Bit	Field	Bits	Definition	Comments
				1: EIGHTTAP_SMOOTH 2: EIGHTTAP_SHARP 3: BILINEAR	
11	31:26	Reserved	6	MBZ	
12..15	31:0	Reserved		MBZ	

Allowed SB Size Encoder Only

The following table details the SB size allowed and the number of records per SB for the encoder.

Allowed SB Size – Encoder Only

SB Size Allowed	Number of Records per SB
64x64	64

Note: HW will support partial SBs within a frame bourndary to a minimun CU8x8 granularity

HCP Commands

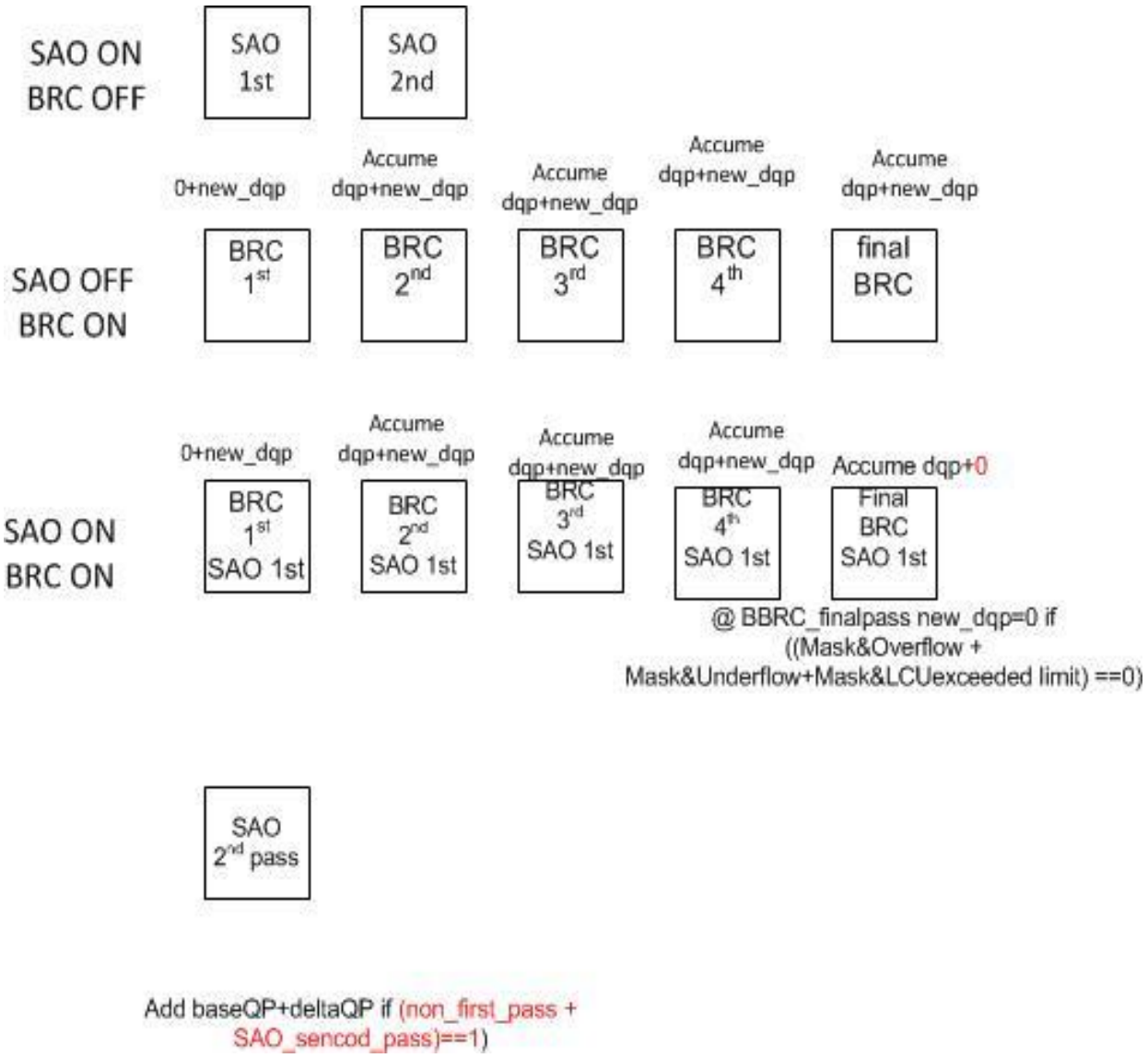
The HCP Commands specify the HEVC BSD object and PAK object level configuration.

HCP_BSD_OBJECT (VideoCS)

HCP_PAK_OBJECT_VideoCS

HCP_PAK_INSERT_OBJECT_VideoCS

Multipass flow during BRC and SAO



CU and Slice level stat streamOut

Along with final bitstream, HLC writes out two statistics related streamout cachelines to the memory. Streamout0 cacheline is composed of 4 quarter cachelines, each containing information on CU skip flag, coding block flag for the TUs in a CU, residual/coefficient bit count for a CU, total bit count for CU, LCU exceed limit flag. A typical streamout0 cacheline, therefore, has information on statistics for 4 CUs and lcu exceed limit flag.

Streamout1 cacheline is composed of quarter cachelines., each quarter cacheline consisting of bit count of current slice.

Pak pipeline streamout enable bit, set by HCP_PIPE_MODE_SELECT command, enables or disables the streamout.

Streamout 0: Per CU Quarter Cacheline Format

Level	Field	Width	Cacheline	Comment
CU	CU Skip Flag	1	qcacheline[0]	Packed in Quarter Cacheline in CU format
LCU	LCU exceed limit	1	qcacheline[1]	Packed in Quarter Cacheline in CU format (valid in last CU of LCU)
	Reserved	14	qcacheline[15:2	Reserved
CU	TU CBF Y/U/V	48	qcacheline[63:16]	Packed in Quarter Cacheline in CU format
CU	CU Coefficient Bit Count (Only residual)	18	qcacheline[81:64]	Packed in Quarter Cacheline in CU format
CU	CU Bit Count (all CU Syntax)	18	qcacheline[113:96]	Packed in Quarter Cacheline in CU format
	Reserved	14	qcacheline[127:114]	Reserved

Streamout 1: Per Slice Quarter Cacheline

Level	Field	Width	Cacheline	Comment
Slice	Slice Bit Count (slice header + data + tail)	32	cacheline[31:0]	
	Reserved	32	cacheline[63:32]	
	SlicePositionX[15:0]	16	cacheline[79:64]	
	SlicePositionY[15:0]	16	cacheline[95:80]	
	Reserved	32	cacheline[127:96]	

Definition of the CU Record Structure for Ext Interface – Encoder Only

The following table defines the CU record structure as indirect data to the PAK Object Command. Entries are DW based (4 bytes) and cache aligned. This memory surface is pointed to by the HCP Indirect CU Object Base Address in the HCP_IND_OBJ_BASE_ADDR_STATE Command.

CU Record Structure Definition

DWord	Bitfield	Field	Bits	Definition	Comments
0	1:0	cu_size	2	0: 8x8	
				1: 16x16	
				2: 32x32	
				3: 64x64	
	2	cu_pred_mode	1	0: Intra	For I slices, pred_mode is always 0.
				1: Inter	
	3	cu_transquant_bypass_flag	1		Note: HW ignores this bit for RhoDomain calculation so the statics will be slightly inaccurate. This bit should not be set to 1 if transform_skip=1
	6:4	cu_part_mode	3	0: 2Nx2N	
				1: 2NxN (inter only)	
				2: Nx2N (inter only)	
				3: NxN (intra only)	only if CU size is 8x8
				4: 2NxhN (inter only)	
				5: hNx2N (inter only)	
	7	IPCM_enable	1	1: Enable IPCM 0: Disable IPCM	MBZ Reserved If IPCM is enabled, then entire CU is IPCM predicted. Both PU and TU sizes should be same as CU size. Cu_pred_mode is ignored when IPCM is enbled. Note: Supports 8bit pixel depth only
	10:8	intra_chroma_mode	3	0: DM	(use Luma mode, from block 0 if NxN)
				1: Reserved	(supposedly to be defined for LM mode)

DWord	Bitfield	Field	Bits	Definition	Comments
				2: Planar	
				3: Vertical	
				4: Horizontal	
				5: DC	
	11	zero_out_coefficient	1	0: Do not force coefficients to zero 1: Force coefficients to zero	If this bit it set to 1, HW will force coefficients to zero.
	15:12	Reserved	5		
	22:16	cu_qp	7	in 7-bit	Valid range: 0 to 51 for 8bit mode -12 to 51 for 10bit mode *diff_cu_qp_delta_depth = 0 (No QP change allowed at CU Level.* Only allow QP change across LCU, no change across CU for PAK only.
	23	cu_qp_sign	1	0:positive 1:negative	Indicates sign bit for QP. Must be zero for 8bit mode
	31:24	interpred_idc[3:0][1:0]	8	2 bits each 0: L0 1: L1 2: Bi 3: reserved	in Z-order interpred_idc[0][1:0] - block 0 = [25:24] interpred_idc[15][1:0] - block 15 = [31:30]
1	5:0	intra_mode[0][5:0]	6	final explicit luma mode. Valid values are 0..34.	1 per cu partition, and only active partitions have valid intra mode value
	7:6	Reserved	2		
	13:8	intra_mode[1][5:0]	6	final explicit luma mode. Valid values are 0..34.	
	15:14	Reserved	2		
	21:16	intra_mode[2][5:0]	6	final explicit luma mode. Valid values are 0..34.	
	23:22	Reserved	2		
	29:24	intra_mode[3][5:0]	6	final explicit luma mode. Valid values are	

DWord	Bitfield	Field	Bits	Definition	Comments
				0..34.	
	31:30	Reserved	2		
2	15:0 31:16	mvx_l0[0][15:0] mvx_l0[1][15:0]	64	16-bit each	[0] in the least sig position in Z-order (1st index starts at LSB and goes up)
3	15:0 31:16	mvx_l0[2][15:0] mvx_l0[3][15:0]			
4	15:0 31:16	mvy_l0[0][15:0] mvy_l0[1][15:0]	64	16-bit each	in Z-order (1st index starts at LSB and goes up)
5	15:0 31:16	mvy_l0[2][15:0] mvy_l0[3][15:0]			
6	15:0 31:16	mvx_l1[0][15:0] mvx_l1[1][15:0]	64	16-bit each	in Z-order (1st index starts at LSB and goes up)
7	15:0 31:16	mvx_l1[2][15:0] mvx_l1[3][15:0]			
8	15:0 31:16	mvy_l1[0][15:0] mvy_l1[1][15:0]	64	16-bit each	in Z-order (1st index starts at LSB and goes up)
9	15:0 31:16	mvy_l1[2][15:0] mvy_l1[3][15:0]			
10	3:0 7:4 11:8 15:12	reserved[0] intra_chroma_mode1[2:0] reserved[0] intra_chroma_mode2[2:0] reserved[0] intra_chroma_mode3[2:0] reserved[3:0]		3-bits each	
10	3:0 7:4 11:8 15:12	ref_idx_l0[0][3:0] ref_idx_l0[1][3:0] ref_idx_l0[2][3:0] ref_idx_l0[3][3:0]	16	4-bit each	in Z-order (1st index starts at LSB and goes up)
	19:16 23:20 27:24 31:28	ref_idx_l1[0][3:0] ref_idx_l1[1][3:0] ref_idx_l1[2][3:0] ref_idx_l1[3][3:0]	16	4-bit each	Combined list is not supported in Z-order (1st index starts at LSB and goes up)
11		tu_size[15:0]	32	0: 4x4	in Z-order (1st index starts at LSB and goes up)
				1: 8x8	
				2: 16x16	
				3: 32x32	
12	15:0	tu_xform_Yskip[15:0]	16	0: TU transform skip flag for luma	In Z-order (1st index starts at LSB and goes up). Populated for each TU even if transform skip

DWord	Bitfield	Field	Bits	Definition	Comments
				component is not set (normal transform) 1: TU transform skip flag for luma component is set	is not supported for that particular TU size. This bit should not be set to 1 if transquant_bypass=1
	27:16	Reserved	12		
	31:28	tu_countm1[3:0]	4	number of TU count per CU minus 1	Intel restriction max 16 TU per CU (however spec allows up to 256 TUs). If there is no TU inside a CU, it is indicated by cbf and skip flag.
13	15:0	tu_xform_Uskip[15:0]	16	0: TU transform skip flag for chroma cb component is not set (normal transform) 1: TU transform skip flag for chroma cb component is set	Indexed by tu_xform_Yskip index. Populated for each TU even if transform skip is not supported for that particular TU size. For the case where a 4x4 chroma TU is associated with a group of four 4x4 luma TUs the chroma cb transform skip is coded in the same position as the last 4x4 luma TU (the prior 3 indexed positions are skipped/ignored). For example an 8x8 CU with four 4x4 luma TUs would code the chroma cb TU transform skip flag as tu_xform_Uskip[3]. Restriction: For 422, 2 TUs for each Chroma correspond to 1 Luma TU share the same transformskip flag. This bit should not be set to 1 if transquant_bypass=1
	31:16	tu_xform_Vskip[15:0]	16	0: TU transform skip flag for chroma cr component is not set (normal transform) 1: TU transform skip flag for chroma cr component is set	Indexing identical tu_xform_Uskip. Populated for each TU even if transform skip is not supported for that particular TU size. This bit should not be set to 1 if transquant_bypass=1

Intel restriction max 16 TU per CU, max 256 TUs in a CU.

Max 64 CUs, and each CU record max is 1 cacheline 64 bytes in size (1 cacheline 64 bytes)

LCU, CU, TU, and PU Sizes – Encoder Only

LCU/CU Partitioning Configurations

LCU size	min CU size	CU Depth	Hierarchical Depth=CU Depth+1
64x64	64x64	0	1
	32x32	1	2
	16x16	2	3
	8x8	3	4
32x32	32x32	0	1
	16x16	1	2
	8x8	2	3
16x16	16x16	0	1
	8x8	1	2
8x8	8x8	X	Not allowed in spec

PU Options for a Given CU

Current CU size (leaf node)	min CU sizes (Pic State)	Allowed PU partition types.
64x64 (2Nx2N) Must be a LCU	64x64	Skip : 2Nx2N Intra : 2Nx2N, NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	32x32	Skip : 2Nx2N Intra : 2Nx2N (no NxN defined in the spec.) Inter :

		2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	16x16	Skip : 2Nx2N Intra : 2Nx2N (no NxN defined in the spec.) Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	8x8	Skip : 2Nx2N Intra : 2Nx2N (no NxN defined in the spec.) Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
32x32 (2Nx2N) Can or is not a LCU	32x32	Skip : 2Nx2N Intra : 2Nx2N,

		NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	16x16	Skip : 2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	8x8	Skip : 2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
16x16 (2Nx2N) Can or is not a LCU	16x16	Skip : 2Nx2N Intra : 2Nx2N, NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	8x8	Skip :

		2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
8x8 (2Nx2N) Cannot be a LCU	8x8	Skip : 2Nx2N Intra : 2Nx2N and NxN Inter : 2Nx2N, 2NxN, Nx2N (both NxN and AMP are not allowed for 8x8 inter CU)

Note: In an 8x8 Inter CU NxN isn't allowed if the SPS parameter disable_inter_4x4 is 1. In Main profile currently this flag is always 1.

U.D, L and R (Up, Down, Left and Right)

n = ¼ or – .

TU Partitioning for a Given CU

CU size	TU size	TU Depth	Max Depth=TU Depth+1	PAK supported TU sizes and corresponding number of TUs in CU
64x64	64x64	0	1	no 64x64 transform, so automatically breakdown into 4 32x32 TUs.
	32x32	1	2	number of TUs in CU = 4
	16x16	2	3	number of TUs in CU = 16
	8x8	3	4	this configuration is currently not supported.

	4x4	4	5	this configuration is currently not supported.
32x32	32x32	0	1	number of TUs in CU = 1
	16x16	1	2	number of TUs in CU = 4
	8x8	2	3	number of TUs in CU = 16
	4x4	3	4	this configuration is currently not supported.
16x16	16x16	0	1	number of TUs in CU = 1
	8x8	1	2	number of TUs in CU = 4
	4x4	2	3	number of TUs in CU = 16
8x8	8x8	0	1	number of TUs in CU = 1
	4x4	1	2	number of TUs in CU = 4

The actual level of partitioning is governed by

- MaxTUSize and MinTUSize in Pic State.
- max_transform_hierarchy_depth_inter <= 2 (intel restriction) DW4 bit 3:2 Pic State
- max_transform_hierarchy_depth_intra <= 2 (intel restriction) DW4 bit 1:0 Pic State

Allowed LCU Size – Encoder Only

The following table details the LCU size allowed and the number of records per LCU for the encoder.

LCU Size Allowed	Fixed Number of Records per LCU
64x64	64
32x32	16
16x16	4

Note: 0.5 CL per CU record in VME mode and 1 CL per CU record in extENC mode.

HEVC Error Concealment

The HCP implements an error concealment policy, which is always enabled and cannot be disabled. The objective is that the HCP will always complete a frame/field workload by either decoding the bit stream normally until it finishes the workload or by concealing blocks until the slice or workload is completed. It should never be allowed to hang.

Error concealment, implemented by the HCP hardware, is configured for each slice in the HCP_BSD_OBJECT command. The following information in the HCP_BSD_OBJECT command is utilized for error concealment.

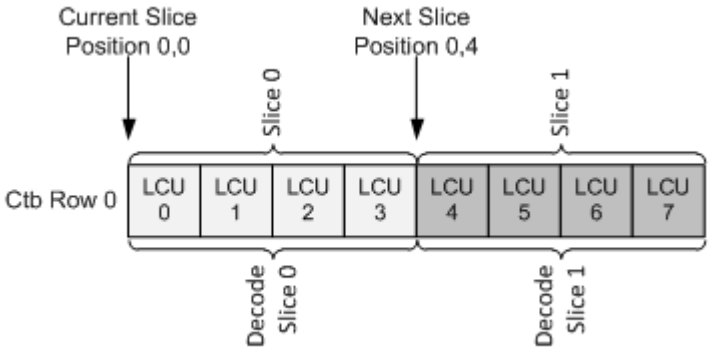
- **SliceStartCtbY, SliceStartCtbX:** The current slice position specified in Ctb coordinates.
- **NextSliceStartCtbY, NextSliceStartCtbX:** The next slice position specified in Ctb coordinates. If the current slice is the last slice in the picture, the next slice values are set to (0,0).
- **LastSliceofPic:** Indicates that the current slice is the last slice in the picture.
- **slice_type:** Indicates the picture type: I, P or B.

The host software will remove all extra slices in the picture. The HCP will not be given a workload that includes extra slices beyond the picture. The last slice in the picture will always be marked by the host software.

The host software will remove any overlapping slices in the picture. The HCP will not be given a workload that includes overlapping slices in the picture.

A HCP_BSD_OBJECT command will include the current slice position and the next slice position. For non-errored streams, it is guaranteed that the slice bit stream will be decoded by the HCP starting from the current slice position through to the Ctb (inclusive) adjacent to the Ctb indicated by the next slice position. *HEVC Error Concealment* illustrates the example of a non-errored stream decode starting with XXX.

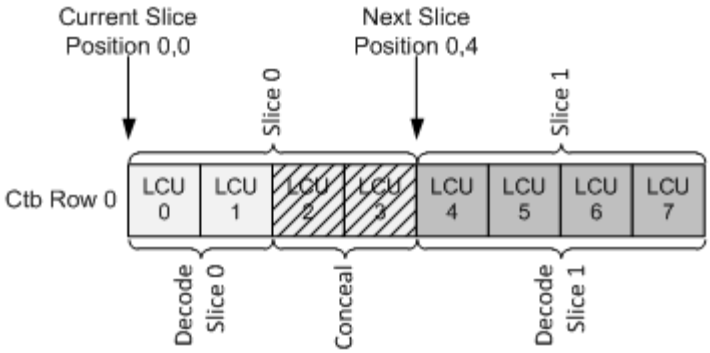
HEVC Slice Decode for Non-errored Stream Cases



For error stream cases where the next slice position does not align itself with the last successfully decoded Ctb in the current slice, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb prior to the Ctb indicated by the next slice position. If the error occurs such that the current decoded Ctb cannot be decoded, the HCP will ensure that the current Ctb is written out by

any means before writing out concealed Ctbs for the remaining Ctbs in the current slice. In the case of the last slice in a picture, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb position in the picture indicated by the resolution of the picture in the HCP_PICT_STATE command. *HEVC Error Concealment* illustrates the case described.

HEVC Slice Decode for Missing Blocks in a Slice



Since the host software removes overlapping slices, the next slice position will never be equal to or less than the current slice position.

A concealed Ctb for an I-slice is constructed by the HCP specifying the Intra_Planar prediction mode for the Ctb.

A concealed Ctb for a P-slice is constructed by the HCP specifying the skip_flag.

A concealed Ctb for a B-slice is constructed by the HCP specifying the skip_flag.

VP9 Register Definitions

This section describes the VP9 Register Definitions as follows:

- Register Attributes Description
- VP9 Register Map
- VP9 Encoder Register Descriptions

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeroes.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

VP9 Register Map

These are MMIO register definitions for decoder/encoder:

VP9 Encoder Register Descriptions

HCP_IMAGE_STATUS_MASK - HCP Image Status Mask

HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control

HCP_UNIT_DONE - HCP Unit Done

HEVC Register Definitions

The Message Channel Interface is a read-only bus used to access the HCP status registers. All registers are 32 bits where reserved bits return a value of zero and subtractive-decode is used to return 0x0000 for all register holes. The Unit ID is 28h. For HCP, the address range is 0x0001E900h to 0001E9FFh.

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeros.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

HCP Decoder Register Map

This documents all HEVC Decoder MMIO Registers.

HCP Decoder Register Descriptions

The HCP implements the following MMIO registers. A description of the register including its address and DWord descriptions are provided.

HCP CABAC Status

HCP Picture Checksum cIdx0

HCP Picture Checksum cIdx1

HCP Picture Checksum cIdx2

HCP Encoder Register Map

These are MMIO register definitions for encoder.

HCP Encoder Register Descriptions

HCP_FRAME_PERFORMANCE_CT - HCP Frame Performance Count

HCP_LAT_CT1 - HCP Memory Latency Count1

HCP_LAT_CT2 - HCP Memory Latency Count2

HCP_LAT_CT3 - HCP Memory Latency Count3

HCP_LAT_CT4 - HCP Memory Latency Count4

HCP_LAT_CT5 - HCP Memory Latency Count5

HCP_LAT_CT6 - HCP Memory Latency Count6

HCP_BIN_CT - HCP Frame BitStream BIN Count

HCP_READ_CT - HCP Frame Motion Comp Read Count

HCP_MISS_CT - HCP Frame Motion Comp Miss Count

HCP_BITSTREAMSE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only

HCP_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register

HCP_CABAC_INSERTION_COUNT - Reported Bitstream Output CABAC Insertion Count

HCP_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register

HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control

HCP_QP_STATUS_COUNT - HCP Qp Status Count

HCP_UNIT_DONE - HCP Unit Done

HCP_IMAGE_STATUS_MASK - HCP Image Status Mask

HUC Status Register Descriptions

Following are HUC Status Register Descriptions:

UKERNEL_HDR_INFO - uKernel Header Info

Acronyms and Applicable Standards

Acronyms and Abbreviations

The table below defines acronyms and abbreviations used in this document.

Acronyms

Acronym	Meaning
AAC	Advanced Audio Coding — part of the MPEG specification, AAC is the latest development in audio compression. It provides higher-quality audio reproduction than MPEG-1 Layer 3 (MP3), while requiring nearly 50% less data. It is defined in ISO/IEC 13818-7.
ADSL	Asymmetrical Digital Subscriber Line — an asymmetrical DSL technology that takes advantage of the one-way nature of most multimedia communication, and provides much faster data rates for downstream (to the subscriber) then the upstream.
API	Application Programming Interface — a set of routines used by an application program to request and carry out low-level services performed by the operating system.
ARGB	Alpha Red Green Blue — color channel components.
ARIB	Association of Radio Industries and Business — designated by the Ministry of Public Management, Home Affairs, Posts and Telecommunications (MPHPT) in Japan. ARIB members include broadcasters, radio equipment manufacturers, telecommunication operators, and related organizations.
ASP	Advanced Simple Profile – MPEG4-2
ATSC	ATSC Advanced Television Systems Committee – an organization in US that establishes and promotes technical standards for advanced television systems, such as digital television (DTV).

Acronym	Meaning
BDU	Bit-stream Data Unit
BIST	Built In Self Test
BPP	Bits Per Pixel
BSD	Byte Stream Decoder
CA, CAM	Conditional Access, Conditional Access Module – the removable descrambling module implemented in digital cable or satellite television system. The data flows through the module, which can have any proprietary scrambling algorithm implemented, yet maintaining system interface compatibility. The CAMs are usually provided by the operators in the TV network.
CPU	Central Processing Unit
DAA	Direct Access Arrangement
DAC	Digital-to-Analog Converter
DDA	Digital Difference Analyzer
DDS	Direct Digital Synthesizer
DPB	Decoded Picture Buffer. This buffer holds the decoded pictures for reference and for output along with the currently decoding picture. This differs from the DPB in the standard, which only holds the decoded pictures for reference.
DVB	Digital Video Broadcasting — a set of open worldwide standards that define digital broadcasting using existing satellite, cable, and terrestrial infrastructures. It uses MPEG-2 specification as a universal foundation and expands it with DVB data structures and processes DVB-compliant digital broadcasting and equipment is widely available to consumers and is indicated with the DVB logo.
DVB-S	Satellite television DVB standards, based on QPSK and 8-DPSK modulation.
DVB-T	Terrestrial television DVB standards, based on 2k and 8k OFDM modulation.
DVD	Digital Versatile Disc
DVD-R	Recordable DVD. Since different disk formats are currently in use, including DVD-R,DVD+R, they are collectively mentioned as DVD-R in this document
DVI	Digital Visual Interface standard (EIA/CEA-861A). The standard defines a method for sending digital video signals over DVI and OpenLDI interface specifications. The standard is fully backward compatible with earlier DVI standards. New features include carrying auxiliary video information, such as aspect ratio and native video format information.
DVO	Digital Video Output - the parallel, low voltage signaling interface defined for Intel® video chipsets
DSL xDSL	Digital Subscriber Line – transmission of data over copper telephone lines capable of bringing high-bandwidth to subscribers. Many flavors of DSL are currently in use, which are collectively called xDSL throughout the document.
DSP	Digital Signal Processor
DST	Destination
DWord	A 32-bit word
ES	Elementary Streams — the raw output of an encoder, containing only what is necessary for a decoder to approximate the original picture or audio.
FIFO	First in First Out

Acronym	Meaning
FIR	Finite Impulse Response
FPU	Floating Point Unit
FW	Firmware running on the decoder controller, as used in Volume 4 of the *Olo River Plus Silicon EAS*
IDR	Instantaneous Decoding Refresh
IEEE 1394 1394	IEEE 1394 or iLink* or FireWire* An IEEE electronics industry standard for connecting multimedia and computing Up to 63 devices can be attached to your PC via a single plug-and-socket connection.
IEEE 802.11 802.11	The Institute for Electronics and Electrical Engineers (IEEE) wireless network specification. 802.11g and 802.11a networks can transmit payload at the rates in excess 34Mbits/s and allow for the wireless transmission at distances from several dozen to several hundred feet indoors.
IF	Intermediate Frequency — the fixed, relatively low-frequency carrier to which current programs are ported by the tuner.
GMCH	Graphics and Memory Control Hub — a chip that connects the IA processor to memory and other components in PC.
HDD	Hard Disk Drive — magnetic mass storage device used in media centers for audiovisual program recording.
HDMI	High Definition Multimedia Interface (HDMI). This interface is used between any audio/video source, such as a set-top box, DVD player, or A/V receiver, and an audio or video monitor, such as a DTV. HDMI supports standard, enhanced or high-definition video, plus multi-channel digital audio on a single cable. The format transmits all ATSC HDTV standards and supports eight-channel digital audio (at up to a 192kHz sampling rate), with bandwidth to spare for future enhancements.
HDTV	High-Definition Television — HDTV specifically refers to the highest-resolution formats of the 18 total DTV formats, true HDTV is generally considered to be 1,080-line interlaced (1080i) or 720-line progressive (720p).
HSR	Hidden Surface Removal
HW	Hardware
I/F	Interface
IEEE	IEEE 32-bit Floating Point number format representation
ISP	Image Synthesis Processor — A collective term to describe all components of the hidden surface removal operation within the PowerVR architecture.
LOD	Level Of Detail — used in texturing calculations.
LSB	Least Significant Bit
LUT	Look-up table
MBAFF	Block Adaptive Field Frame mode
MFD	Multi-Format Decoder
MMU	Memory Management Unit
MMMC	Multi-port, Multi-channel Memory Controller
MSA	Intel Micro Signal Architecture — microprocessor architecture combining the features of microcontroller and digital signal processor. MSA is used here as a synonym of the processor core

Acronym	Meaning
	used in Olo River Plus
MSB	Most Significant Bit
MPEG	Motion Picture Experts Group – Organization that develops standards for digital video and digital audio compression.
MPR	Inter Prediction Module
NAL	Network Abstraction Layer
NAL unit	Syntax structure in a H.264 stream
NTSC	National Television System Committee, North American 525-line analog broadcast TV standard.
NIM	Network Interface Module – the integrated tuner and digital demodulator in the (satellite) TV systems. The DVB NIMs output MPEG transport stream.
NOP	No operation
OEM	Original Equipment Manufacturer
OGL/OpenGL	Open GL application programming interface
PAL	Phase Alternation Line - TV standard used in Europe. PAL uses 625 lines per frame, a 25 frames per second update rate and YUV color encoding. The number of visible pixels for PAL video is 768 x 576.
PCI	Peripheral Component Interconnect bus, a bi-directional bus defined in PCI 2.x specification
PES	Packetized Elementary Streams — packetized streams are the ES streams arranged in data packets with PES header starting every packet. The syntax of the ES and PES is defined in MPEG. See definition for ES.
PIP	Picture In Picture display mode
POD	Point of Deployment conditional access module — the removable conditional access module defined in the OpenCable* specification in US.
PPS	Picture Parameter set
PTS	Presentation time stamp
PVR	Personal Video Recorder, also PDR or personal digital recorder — an interactive TV-recording device that records programs in digital format and allows users to search for/record shows based on type (for instance all basketball games or all episodes of a particular program). Users can also pause, rewind, stop, or fast-forward live programs with only a small time lag.
PWL	Piece-wise Linear
PXD	Pixel Decoder Module
RF	Radio Frequency – usually, modulated carriers which can be directly received by the tuners of TVs or radio receivers
RISC	Reduced Instruction Set Computer
RHW	Reciprocal Homogenous W — W is a 3-D coordinate representation like X Y Z
RSB	Row Store Buffer
RTL	Register Transfer Language/Level
SEI	Supplementary Enhancement Information

Acronym	Meaning
SIF	Semaphore Interface Module
SIMD	Single Instruction Multiple Data
SMPTE	Society of Motion Picture and Television Engineers
SOC	System on chip
SP	Simple Profile – MPEG4-2
SPS	Sequence Parameter set
SRC	Source
SDTV	Standard-Definition Television — a digital television system that is similar to current analog TV standards in picture resolution and aspect ratio. Typical SDTV resolution is 480i or 480p.
STB	Set Top Box — a device that effectively turns a television set into an interactive Internet device and/or allows the television to receive and decode digital television (DTV) broadcasts.
TA	Tile Accelerator
TS	MPEG-2 Transport Stream — a sequence of 188-byte packets carrying the multi-program audiovisual data
TSP	Texture Shading Processor — a collective term to describe all components of the texture, shading and pixel blending operations within the PowerVR architecture.
VCL	Video Coded Layer
VCXO	Voltage Controlled Crystal Oscillator
VGP/ VGP Lite	Vertex Geometry Processor
VLC	Variable length coded. This refers to the collection of coding techniques that are used in VC1, and include CABAC, CAVLC and Exp-Golomb.
VOL	Video Object Layer
VOP	Video Object Plane
WAN	Wide Area Network
WSS	Wide Screen Signaling
XDS	Extended Data Services — data services sending data in line 21/283 of the analog NTSC TV signal
XSI	Intel® XScale® System Interconnect
X, Y, Z, W	3-D coordinate representations
YUV	YUV texture format, primarily for video formats

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 11: Blitter

Janaury 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Blitter (BLT) Engine Command Streamer 1
Software Control Bit Definitions 1
Registers for Blitter Engine Command Streamer 1
BLT Engine 1
Introduction 1
Classical BLT Engine Functional Description 2
Basic BLT Functional Considerations 2
Color Depth Configuration and Color Expansion 2
Graphics Data Size Limitations 3
Bit-Wise Operations 3
Per-Pixel Write-Masking Operations 8
When the Source and Destination Locations Overlap 8
Basic Graphics Data Considerations 12
Contiguous vs. Discontinuous Graphics Data 12
Source Data 14
Monochrome Source Data 14
Pattern Data 15
Destination Data 18
BLT Programming Examples 19
Pattern Fill — A Very Simple BLT 19
Drawing Characters Using a Font Stored in System Memory 22
BLT Instruction Overview 26
BLT Engine State 26
Cacheable Memory Support 27
Device Cache Coherency: Render & Texture Caches 27
BLT Engine Instructions 27
BLT Programming Restrictions 28
BLT Engine Instruction Field Definitions 29

Blitter (BLT) Engine Command Streamer

A separate pipeline for Blitter operations is used in future generations. This pipeline has its own command streamer and operates completely independently of the other command streamers. This command streamer supports a separate set of registers starting at offset 20000h.

Software Control Bit Definitions

Registers in the range 22XX are not protected from the load register immediate instruction if the command is executed in the non-secure batch buffer.

BCS_SWCTRL - BCS SW Control

Registers for Blitter Engine Command Streamer

These are the Registers for the Blitter Engine Command Streamer.

Also see the Observability volume for related information.

GAB_MODE - Mode Register for GAB

BLT Engine

Introduction

2D Rendering can be divided into 2 categories: classical BLTs, described here, and 3D BLTs. 3D BLTs are operations which can take advantage of the 3D drawing engine's functionality and access patterns.

Functions such as Alpha BLTs, arithmetic (bilinear) stretch BLTs, rotations, transposing pixel maps, color space conversion, and DIBs are all considered 3D BLTs and are covered in the 3D rendering section. DIBs can be thought of as an indexed texture which uses the texture palette for performing the data translation. All drawing engines have swappable context. The same hardware can be used by multiple driver threads where the current state of the hardware is saved to memory and the appropriate state is loaded from memory on thread switches.

All operands for both 3D and classical BLTs can be in graphics aperture or cacheable system memory. Some operands can be immediates which are sent through the command stream. Immediate operands are: patterns, monochrome sources, DIB palettes, and DIB source operands. All non-monochrome operands which are not tiled have a stride granularity of a double-word (4 bytes).

The classical BLT commands support both linear addressing and X, Y coordinates with and without clipping. All X1 and Y1 destination and clipping coordinates are inclusive, while X2 and Y2 are exclusive. Currently, only destination coordinates can be negative. The source and clipping coordinates must be positive. If clipping is disabled, but a negative destination coordinate is specified, the negative coordinate is clipped to 0. Linear address BLT commands must supply a non-zero height and width. If either height or width = 0, then no accesses occur.

Classical BLT Engine Functional Description

The graphics controller provides a hardware-based BLT engine to off load the work of moving blocks of graphics data from the host CPU. Although the BLT engine is often used simply to copy a block of graphics data from the source to the destination, it also has the ability to perform more complex functions. The BLT engine is capable of receiving three different blocks of graphics data as input as shown in the figure below. The source data may exist in the frame buffer or the Graphics aperture. The pattern data always represents an 8x8 block of pixels that can be located in the frame buffer, Graphics aperture, or passed through a command packet. The pattern data must be located in linear memory. The data already residing at the destination may also be used as an input. The destination data can also be located in the frame buffer or graphics aperture.

Block Diagram and Data Paths of the BLT Engine

The BLT engine may use any combination of these three different blocks of graphics data as operands, in both bit-wise logical operations to generate the actual data to be written to the destination, and in per-pixel write-masking to control the writing of data to the destination. It is intended that the BLT engine will perform these bit-wise and per-pixel operations on color graphics data that is at the same color depth that the rest of the graphics system has been set. However, if either the source or pattern data is monochrome, the BLT engine has the ability to put either block of graphics data through a process called "color expansion" that converts monochrome graphics data to color. Since the destination is often a location in the on-screen portion of the frame buffer, it is assumed that any data already at the destination will be of the appropriate color depth.

Basic BLT Functional Considerations

Color Depth Configuration and Color Expansion

The graphics system and BLT engine can be configured for color depths of 8, 16, and 32 bits per pixel.

The configuration of the BLT engine for a given color depth dictates the number of bytes of graphics data that the BLT engine will read and write for each pixel while performing a BLT operation. It is assumed that any graphics data already residing at the destination which is used as an input is already at the color depth to which the BLT engine is configured. Similarly, it is assumed that any source or pattern data used as an input has this same color depth, unless one or both is monochrome. If either the source or pattern data is monochrome, the BLT engine performs a process called "color expansion" to convert such monochrome data to color at the color depth to which the BLT engine has been set.

During "color expansion" the individual bits of monochrome source or pattern data that correspond to individual pixels are converted into 1, 2, or 4 bytes (which ever is appropriate for the color depth to which the BLT engine has been set). If a given bit of monochrome source or pattern data carries a value of 1, then the byte(s) of color data resulting from the conversion process are set to carry the value of a specified foreground color. If a given bit of monochrome source or pattern data carries a value of 0, the resulting byte(s) are set to the value of a specified background color or not written if transparency is selected.

The BLT engine is set to a default configuration color depth of 8, 16, or 32 bits per pixel through BLT command packets. Whether the source and pattern data are color or monochrome must be specified using command packets. Foreground and background colors for the color expansion of both monochrome source and pattern data are also specified through the command packets. The source foreground and background colors used in the color expansion of monochrome source data are specified independently of those used for the color expansion of monochrome pattern data.

Graphics Data Size Limitations

The BLT engine is capable of transferring very large quantities of graphics data. Any graphics data read from and written to the destination is permitted to represent a number of pixels that occupies up to 65,536 scan lines and up to 32,768 bytes per scan line at the destination. The maximum number of pixels that may be represented per scan line's worth of graphics data depends on the color depth.

Any source data used as an input must represent the same number of pixels as is represented by any data read from or written to the destination, and it must be organized so as to occupy the same number of scan lines and pixels per scan line.

The actual number of scan lines and bytes per scan line required to accommodate data read from or written to the destination are set in the destination width & height registers or using X and Y coordinates within the command packets. These two values are essential in the programming of the BLT engine, because the engine uses these two values to determine when a given BLT operation has been completed.

Bit-Wise Operations

The BLT engine can perform any one of 256 possible bit-wise operations using various combinations of the three previously described blocks of graphics data that the BLT engine can receive as input.

The choice of bit-wise operation selects which of the three inputs will be used, as well as the particular logical operation to be performed on corresponding bits from each of the selected inputs. The BLT engine automatically foregoes reading any form of graphics data that has not been specified as an input by the choice of bit-wise operation. An 8-bit code written to the raster operation field of the command packets chooses the bit-wise operation. The following table lists the available bit-wise operations and their corresponding 8-bit codes.

Bit-Wise Operations and 8-Bit Codes (00-3F)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
00	writes all 0's	20	D and (P and (notS))
01	not(D or (P or S)))	21	not(S or(D xor P))
02	D and (not(P or S))	22	D and (notS)
03	not(P or S)	23	not(S or (P and (notD)))
04	S and (not(D or P))	24	(S xor P) and (D xor S)
05	not(D or P)	25	not(P xor (D and (not(S and P))))
06	not(P or (not(D xor S)))	26	S xor (D or (P and S))

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
07	not(P or (D and S))	27	S xor (D or (not(P xor S)))
08	S and (D and (notP))	28	D and (P xor S)
09	not(P or (D xor S))	29	not(P xor (S xor (D or (P and S))))
0A	D and (notP)	2A	D and (not(P and S))
0B	not(P or (S and (notD)))	2B	not(S xor ((S xor P) and (P xor D)))
0C	S and (notP)	2C	S xor (P and (D or S))
0D	not(P or (D and (notS)))	2D	P xor (S or (notD))
0E	not(P or (not(D or S)))	2E	P xor (S or (D xor P))
0F	notP	2F	not(P and (S or (notD)))
10	P and (not(D or S))	30	P and (notS)
11	not(D or S)	31	not(S or (D and (notP)))
12	not(S or (not(D xor P)))	32	S xor (D or (P or S))
13	not(S or (D and P))	33	notS
14	not(D or (not(P xor S)))	34	S xor (P or (D and S))
15	not(D or (P and S))	35	S xor (P or (not(D xor S)))
16	P xor (S xor (D and (not(P and S))))	36	S xor (D or P)
17	not(S xor ((S xor P) and (D xor S)))	37	not(S and (D or P))
18	(S xor P) and (P xor D)	38	P xor (S and (D or P))
19	not(S xor (D and (not(P and S))))	39	S xor (P or (notD))
1A	P xor (D or (S and P))	3A	S xor (P or (D xor S))
1B	not(S xor (D and (P xor S)))	3B	not(S and (P or (notD)))
1C	P xor (S or (D and P))	3C	P xor S
1D	not(D xor (S and (P xor D)))	3D	S xor (P or (not(D or S)))
1E	P xor (D or S)	3E	S xor (P or (D and (notS)))
1F	not(P and (D or S))	3F	not(P and S)

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (40 - 7F)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
40	P and (S and (notD))	60	P and (D xor S)
41	not(D or (P xor S))	61	not(D xor (S xor (P or (D and S))))
42	(S xor D) and (P xor D)	62	D xor (S and (P or D))

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
43	not(S xor (P and (not(D and S))))	63	S xor (D or (notP))
44	S and (notD)	64	S xor (D and (P or S))
45	not(D or (P and (notS)))	65	D xor (S or (notP))
46	D xor (S or (P and D))	66	D xor S
47	not(P xor (S and (D xor P)))	67	S xor (D or (not(P or S)))
48	S and (D xor P)	68	not(D xor (S xor (P or (not(D or S)))))
49	not(P xor (D xor (S or (P and D))))	69	not(P xor (D xor S))
4A	D xor (P and (S or D))	6A	D xor (P and S)
4B	P xor (D or (notS))	6B	not(P xor (S xor (D and (P or S))))
4C	S and (not(D and P))	6C	S xor (D and P)
4D	not(S xor ((S xor P) or (D xor S)))	6D	not(P xor (D xor (S and (P or D))))
4E	P xor (D or (S xor P))	6E	S xor (D and (P or (notS)))
4F	not(P and (D or (notS)))	6F	not(P and (not(D xor S)))
50	P and (notD)	70	P and (not(D and S))
51	not(D or (S and (notP)))	71	not(S xor ((S xor D) and (P xor D)))
52	D xor (P or (S and D))	72	S xor (D or (P xor S))
53	not(S xor (P and (D xor S)))	73	not(S and (D or (notP)))
54	not(D or (not(P or S)))	74	D xor (S or (P xor D))
55	notD	75	not(D and (S or (notP)))
56	D xor (P or S)	76	S xor (D or (P and (notS)))
57	not(D and (P or S))	77	not(D and S)
58	P xor (D and (S or P))	78	P xor (D and S)
59	D xor (P or (notS))	79	not(D xor (S xor (P and (D or S))))
5A	D xor P	7A	D xor (P and (S or (notD)))
5B	D xor (P or (not(S or D)))	7B	not(S and (not(D xor P)))
5C	D xor (P or (S xor D))	7C	S xor (P and (D or (notS)))
5D	not(D and (P or (notS)))	7D	not(D and (not(P xor S)))
5E	D xor (P or (S and (notD)))	7E	(S xor P) or (D xor S)
5F	not(D and P)	7F	not(D and (P and S))

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (80 - BF)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
80	D and (P and S)	A0	D and P
81	not((S xor P) or (D xor S))	A1	not(P xor (D or (S and (notP))))
82	D and (not(P xor S))	A2	D and (P or (notS))
83	not(S xor (P and (D or (notS))))	A3	not(D xor (P or (S xor D)))
84	S and (not(D xor P))	A4	not(P xor (D or (not(S or P))))
85	not(P xor (D and (S or (notP))))	A5	not(P xor D)
86	D xor (S xor (P and (D or S)))	A6	D xor (S and (notP))
87	not(P xor (D and S))	A7	not(P xor (D and (S or P)))
88	D and S	A8	D and (P or S)
89	not(S xor (D or (P and (notS))))	A9	not(D xor (P or S))
8A	D and (S or (notP))	AA	D
8B	not(D xor (S or (P xor D)))	AB	D or (not(P or S))
8C	S and (D or (notP))	AC	S xor (P and (D xor S))
8D	not(S xor (D or (P xor S)))	AD	not(D xor (P or (S and D)))
8E	S xor ((S xor D) and (P xor D))	AE	D or (S and (notP))
8F	not(P and (not(D and S)))	AF	D or (notP)
90	P and (not(D xor S))	B0	P and (D or (notS))
91	not(S xor (D and (P or (notS))))	B1	not(P xor (D or (S xor P)))
92	D xor (P xor (S and (D or P)))	B2	S xor ((S xor P) or (D xor S))
93	not(S xor (P and D))	B3	not(S and (not(D and P)))
94	P xor (S xor (D and (P or S)))	B4	P xor (S and (notD))
95	not(D xor (P and S))	B5	not(D xor (P and (S or D)))
96	D xor (P xor S)	B6	D xor (P xor (S or (D and P)))
97	P xor (S xor (D or (not(P or S))))	B7	not(S and (D xor P))
98	not(S xor (D or (not(P or S))))	B8	P xor (S and (D xor P))
99	not(D xor S)	B9	not(D xor (S or (P and D)))
9A	D xor (P and (notS))	BA	D or (P and (notS))
9B	not(S xor (D and (P or S)))	BB	D or (notS)
9C	S xor (P and (notD))	BC	S xor (P and (not(D and S)))
9D	not(D xor (S and (P or D)))	BD	not((S xor D) and (P xor D))
9E	D xor (S xor (P or (D and S)))	BE	D or (P xor S)
9F	not(P and (D xor S))	BF	D or (not(P and S))

Notes:

S = Source Data

P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (C0 - FF)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
C0	P and S	E0	P and (D or S)
C1	not(S xor (P or (D and (notS))))	E1	not(P xor (D or S))
C2	not(S xor (P or (not(D or S))))	E2	D xor (S and (P xor D))
C3	not(P xor S)	E3	not(P xor (S or (D and P)))
C4	S and (P or (notD))	E4	S xor (D and (P xor S))
C5	not(S xor (P or (D xor S)))	E5	not(P xor (D or (S and P)))
C6	S xor (D and (notP))	E6	S xor (D and (not(P and S)))
C7	not(P xor (S and (D or P)))	E7	not((S xor P) and (P xor D))
C8	S and (D or P)	E8	S xor ((S xor P) and (D xor S))
C9	not(S xor (P or D))	E9	not(D xor (S xor (P and (not(D and S)))))
CA	D xor (P and (S xor D))	EA	D or (P and S)
CB	not(S xor (P or (D and S)))	EB	D or (not(P xor S))
CC	S	EC	S or (D and P)
CD	S or (not(D or P))	ED	S or (not(D xor P))
CE	S or (D and (notP))	EE	D or S
CF	S or (notP)	EF	S or (D or (notP))
D0	P and (S or (notD))	F0	P
D1	not(P xor (S or (D xor P)))	F1	P or (not(D or S))
D2	P xor (D and (notS))	F2	P or (D and (notS))
D3	not(S xor (P and (D or S)))	F3	P or (notS)
D4	S xor ((S xor P) and (P xor D))	F4	P or (S and (notD))
D5	not(D and (not(P and S)))	F5	P or (notD)
D6	P xor (S xor (D or (P and S)))	F6	P or (D xor S)
D7	not(D and (P xor S))	F7	P or (not(D and S))
D8	P xor (D and (S xor P))	F8	P or (D and S)
D9	not(S xor (D or (P and S)))	F9	P or (not(D xor S))
DA	D xor (P and (not(S and D)))	FA	D or P
DB	not((S xor P) and (D xor S))	FB	D or (P or (notS))
DC	S or (P and (notD))	FC	P or S
DD	S or (notD)	FD	P or (S or (notD))
DE	S or (D xor P)	FE	D or (P or S)
DF	S or (not(D and P))	FF	writes all 1's

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Per-Pixel Write-Masking Operations

The BLT engine is able to perform per-pixel write-masking with various data sources used as pixel masks to constrain which pixels at the destination are to be written to by the BLT engine. As shown in the figure below, either monochrome source or monochrome pattern data may be used as pixel masks. Color pattern data cannot be used. Another available pixel mask is derived by comparing a particular color range per color channel to either the color already specified for a given pixel at the destination or source.

Block Diagram and Data Paths of the BLT Engine

The command packets can specify the monochrome source or the monochrome pattern data as a pixel mask. When this feature is used, the bits that carry a value of 0 cause the bytes of the corresponding pixel at the destination to not be written to by the BLT engine, thereby preserving whatever data was originally carried within those bytes. This feature can be used in writing characters to the display, while also preserving the pre-existing backgrounds behind those characters. When both operands are in the transparent mode, the logical AND of the 2 operands are used for the write enables per pixel.

The 3-bit field, destination transparency mode, within the command packets can select per-pixel write-masking with a mask based on the results of color comparisons. The monochrome source background and foreground are range compared with either the bytes for the pixels at the destination or the source operand. This operation is described in the BLT command packet and register descriptions.

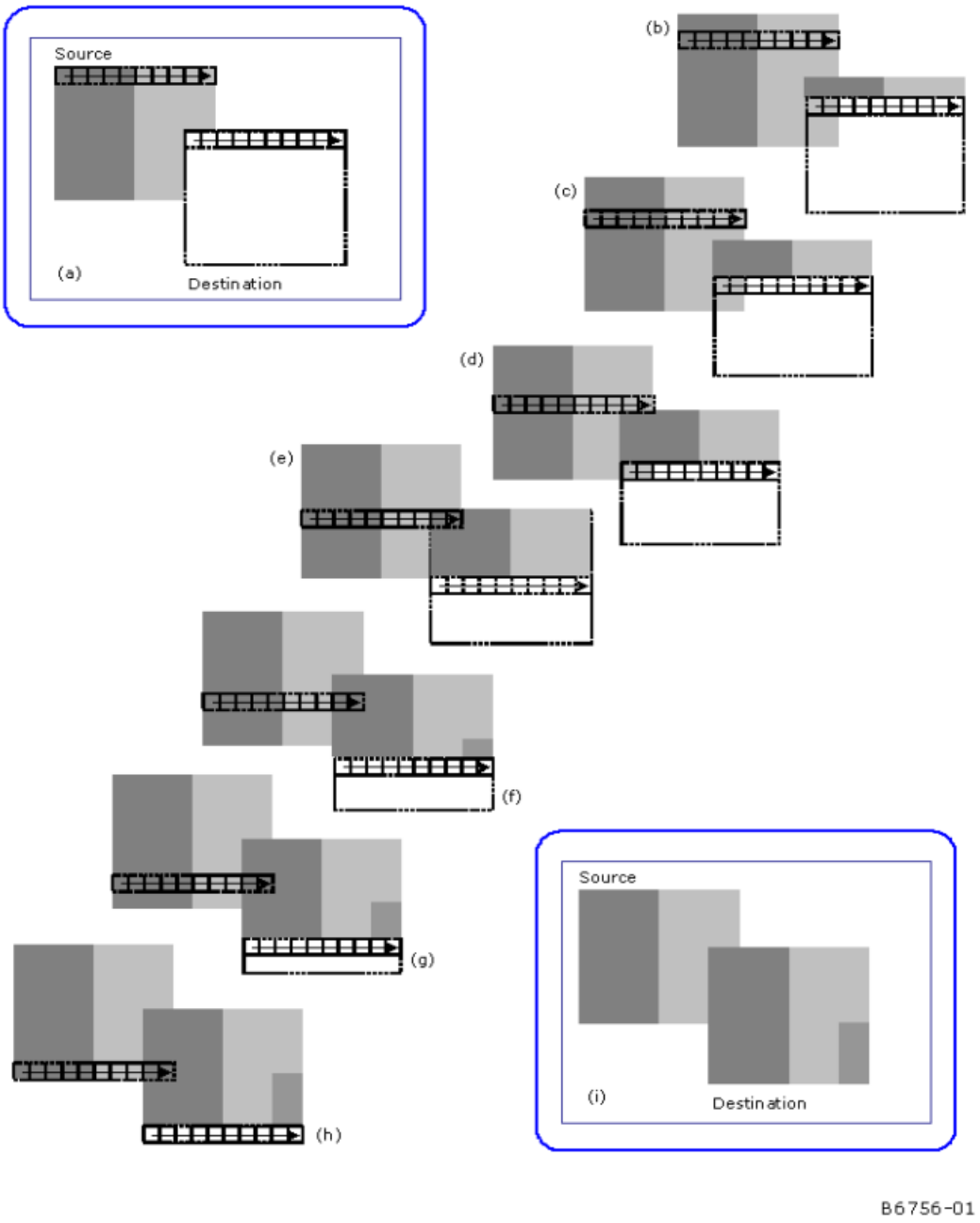
When the Source and Destination Locations Overlap

It is possible to have BLT operations in which the locations of the source and destination data overlap. This frequently occurs in BLT operations where a user is shifting the position of a graphical item on the display by only a few pixels. In these situations, the BLT engine must be programmed so that destination data is not written into destination locations that overlap with source locations before the source data at those locations has been read. Otherwise, the source data will become corrupted. The XY commands determine whether there is an overlap and perform the accesses in the proper direction to avoid data corruption.

The following figure shows how the source data can be corrupted when a rectangular block is copied from a source location to an overlapping destination location. The BLT engine typically reads from the source location and writes to the destination location starting with the left-most pixel in the top-most line of both, as shown in step (a). As shown in step (b), corruption of the source data has already started with the copying of the top-most line in step (a) — part of the source that originally contained lighter-colored pixels has now been overwritten with darker-colored pixels. More source data corruption occurs as steps (b) through (d) are performed. At step (e), another line of the source data is read, but the two

right-most pixels of this line are in the region where the source and destination locations overlap, and where the source has already been overwritten as a result of the copying of the top-most line in step (a). Starting in step (f), darker-colored pixels can be seen in the destination where lighter-colored pixels should be. This errant effect occurs repeatedly throughout the remaining steps in this BLT operation. As more lines are copied from the source location to the destination location, it becomes clear that the end result is not what was originally intended.

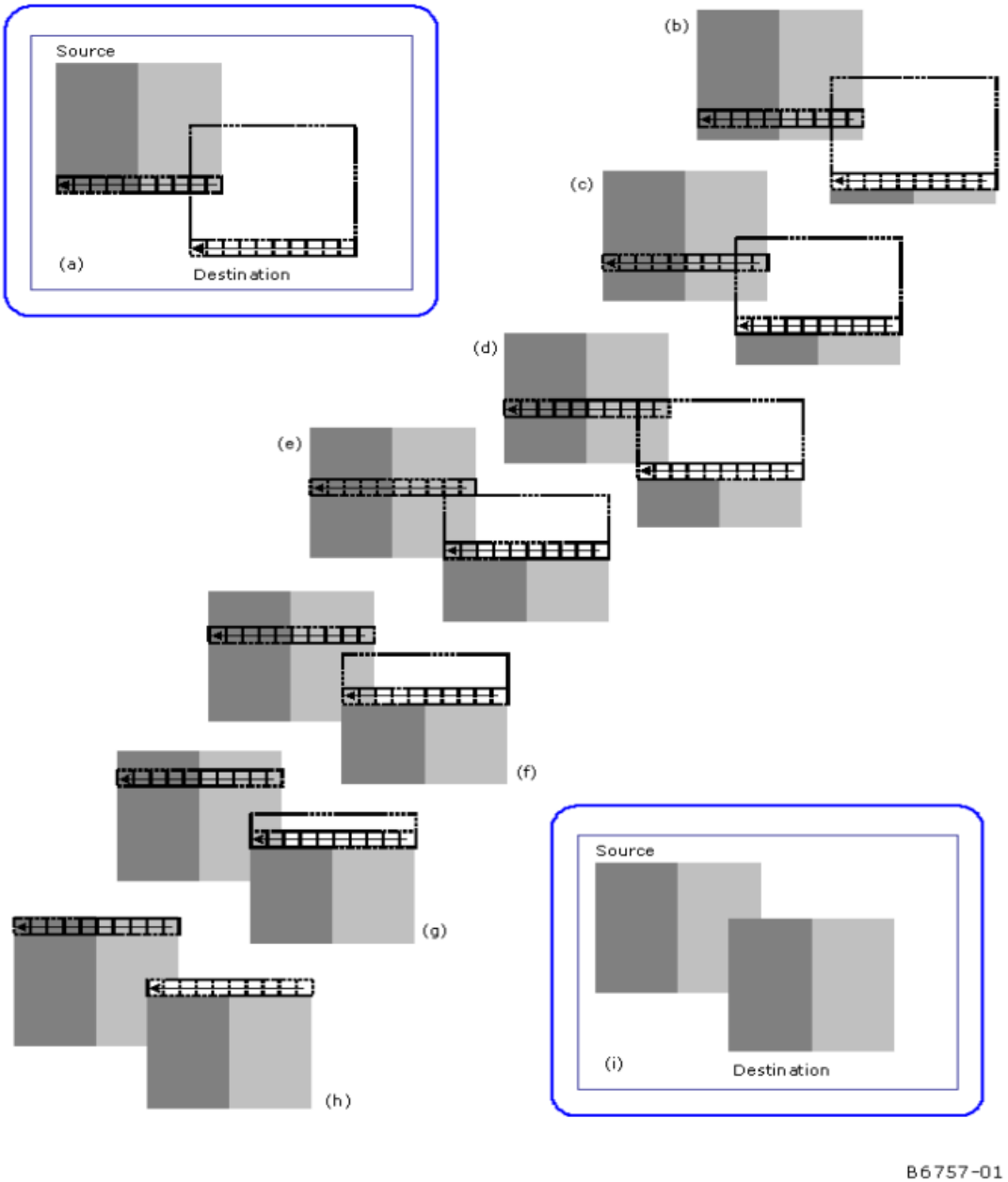
Source Corruption in BLT with Overlapping Source and Destination Locations



The BLT engine can alter the order in which source data is read and destination data is written when necessary to avoid source data corruption problems when the source and destination locations overlap. The command packets provide the ability to change the point at which the BLT engine begins reading and writing data from the upper left-hand corner (the usual starting point) to one of the other three corners. The BLT engine may be set to read data from the source and write it to the destination starting at any of the four corners of the panel.

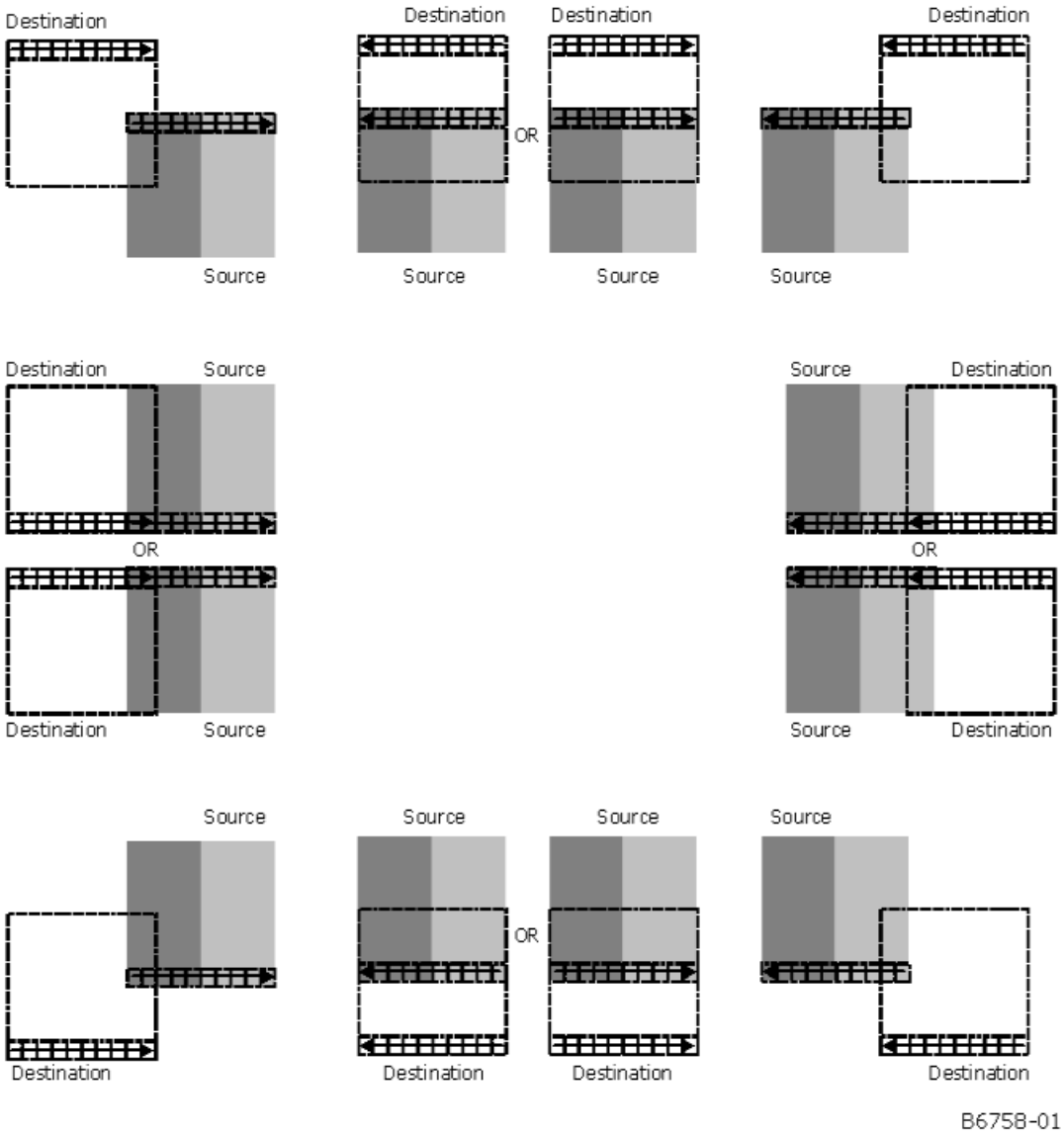
The XY command packets perform the necessary comparisons and start at the proper corner of each operand which avoids data corruption.

Correctly Performed BLT with Overlapping Source and Destination Locations



The following figure illustrates how this feature of the BLT engine can be used to perform the same BLT operation as was illustrated in the figure above, while avoiding the corruption of source data. As shown in the figure below, the BLT engine reads the source data and writes the data to the destination starting with the right-most pixel of the bottom-most line. By doing this, no pixel existing where the source and destination locations overlap will ever be written to before it is read from by the BLT engine. By the time the BLT operation has reached step (e) where two pixels existing where the source and destination locations overlap are about to be over written, the source data for those two pixels has already been read.

Suggested Starting Points for Possible Source and Destination Overlap Situations



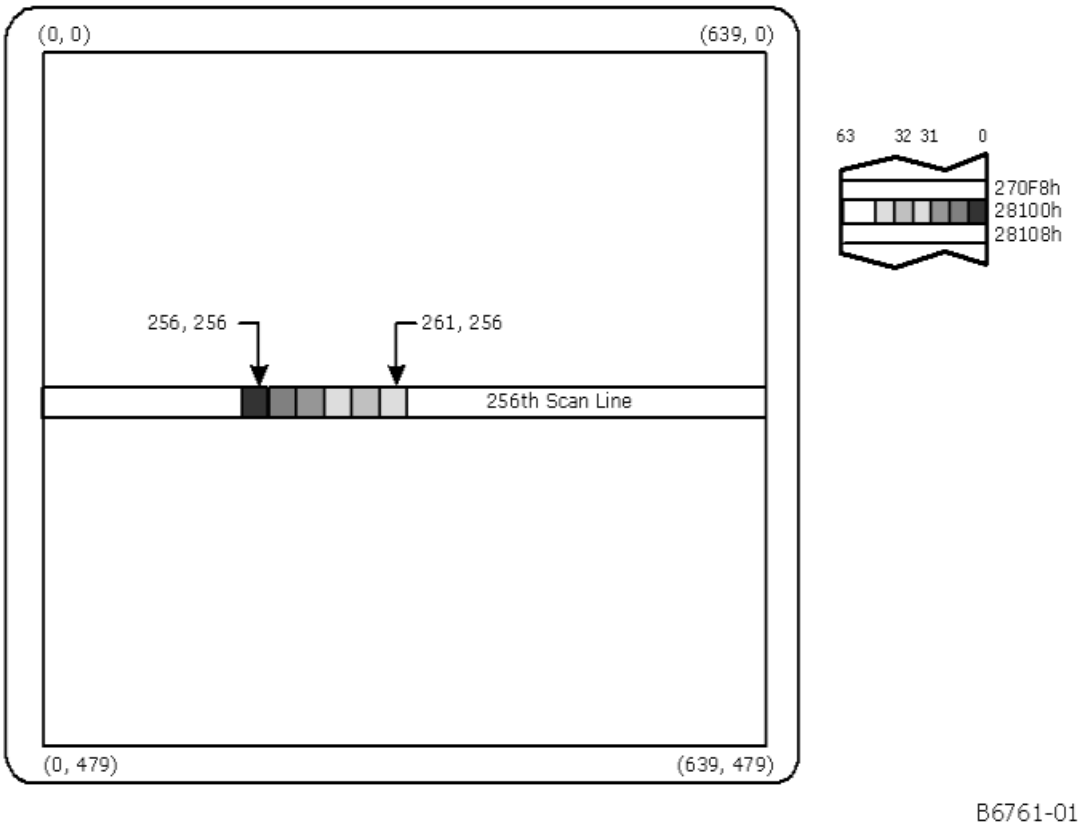
The figure above shows the recommended lines and pixels to be used as starting points in each of 8 possible ways in which the source and destination locations may overlap. In general, the starting point should be within the area in which the source and destination overlap.

Basic Graphics Data Considerations

Contiguous vs. Discontinuous Graphics Data

Graphics data stored in memory, particularly in the frame buffer of a graphics system, has organizational characteristics that often distinguish it from other varieties of data. The main distinctive feature is the tendency for graphics data to be organized in a discontinuous block of graphics data made up of multiple sub-blocks of bytes, instead of a single contiguous block of bytes.

Representation of On-Screen Single 6-Pixel Line in the Frame Buffer

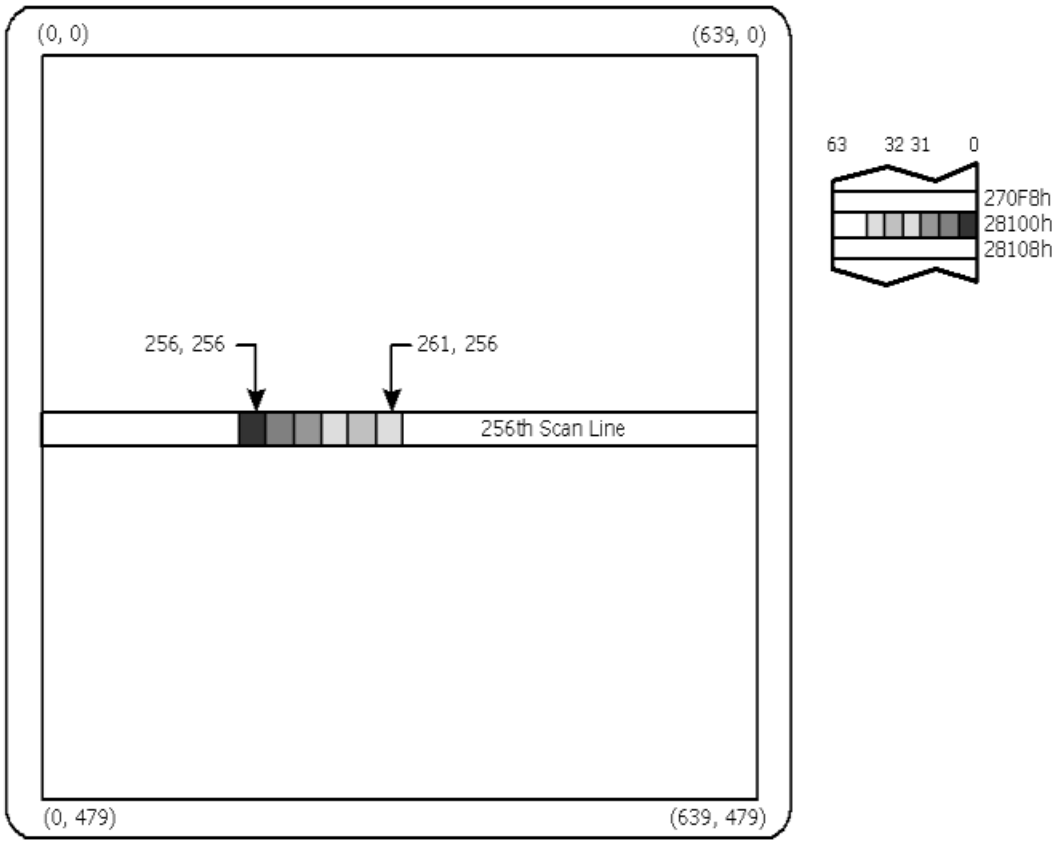


The figure above shows an example of contiguous graphics data — a horizontal line made up of six adjacent pixels within a single scan line on a display with a resolution of 640x480. Presuming that the graphics system driving this display has been set to 8 bits per pixel and that the frame buffer's starting address of 0h corresponds to the upper left-most pixel of this display, then the six pixels that make this horizontal line starting at coordinates (256, 256) occupies the six bytes starting at frame buffer address 28100h, and ending at address 28105h.

In this case, there is only one scan line's worth of graphics data in this single horizontal line, so the block of graphics data for all six of these pixels exists as a single, contiguous block comprised of only these six bytes. The starting address and the number of bytes are the only pieces of information that a BLT engine would require to read this block of data.

The simplicity of the above example of a single horizontal line contrasts sharply to the example of discontinuous graphics data depicted in the figure below. The simple six-pixel line of the figure above is now accompanied by three more six-pixel lines placed on subsequent scan lines, resulting in the 6x4 block of pixels shown.

Representation of On-Screen 6x4 Array of Pixels in the Frame Buffer



Since there are other pixels on each of the scan lines on which this 6x4 block exists that are not part of this 6x4 block, what appears to be a single 6x4 block of pixels on the display must be represented by a discontinuous block of graphics data made up of 4 separate sub-blocks of six bytes apiece in the frame buffer at addresses 28100h, 28380h, 28600h, and 28880h. This situation makes the task of reading what appears to be a simple 6x4 block of pixels more complex. However, there are two characteristics of this 6x4 block of pixels that help simplify the task of specifying the locations of all 24 bytes of this discontinuous block of graphics data: all four of the sub-blocks are of the same length, and the four sub-blocks are separated from each other at equal intervals.

The BLT engine is designed to make use of these characteristics of graphics data to simplify the programming required to handle discontinuous blocks of graphics data. For such a situation, the BLT engine requires only four pieces of information: the starting address of the first sub-block, the length of a sub-block, the offset (in bytes), pitch, of the starting address of each subsequent sub-block, and the quantity of sub-blocks.

Source Data

The source data may exist in the frame buffer or elsewhere in the graphics aperture where the BLT engine may read it directly, or it may be provided to the BLT engine by the host CPU through the command packets. The block of source graphics data may be either contiguous or discontinuous, and may be either in color (with a color depth that matches that to which the BLT engine has been set) or monochrome.

The source select bit in the command packets specifies whether the source data exists in the frame buffer or is provided through the command packets. Monochrome source data is always specified as being supplied through an immediate command packet.

If the color source data resides within the frame buffer or elsewhere in the graphics aperture, then the Source Address Register, specified in the command packets is used to specify the address of the source.

In cases where the host CPU provides the source data, it does so by writing the source data to ring buffer directly after the BLT command that requires the data or uses an IMMEDIATE_INDIRECT_BLT command packet which has a size and pointer to the operand in Graphics aperture.

The block of bytes sent by the host CPU through the command packets must be quadword-aligned and the source data contained within the block of bytes must also be aligned.

To accommodate discontinuous source data, the source and destination pitch registers can be used to specify the offset in bytes from the beginning of one scan line's worth source data to the next. Otherwise, if the source data is contiguous, then an offset equal to the length of a scan line's worth of source data should be specified.

Monochrome Source Data

The opcode of the command packet specifies whether the source data is color or monochrome. Since monochrome graphics data only uses one bit per pixel, each byte of monochrome source data typically carries data for 8 pixels which hinders the use of byte-oriented parameters when specifying the location and size of valid source data. Some additional parameters must be specified to ensure the proper reading and use of monochrome source data by the BLT engine. The BLT engine also provides additional options for the manipulation of monochrome source data versus color source data.

The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome data, the BLT engine converts it into color through a process called color expansion, which takes place as a BLT operation is performed. In color expansion the single bits of monochrome source data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry value corresponding to either the foreground or background color that have been specified for use in this conversion process. If a given bit of monochrome source data carries a value of 1, then the byte(s) of color data resulting from the conversion process will be set to carry the value of the foreground color. If a given bit of monochrome source data carries a value of 0, then the resulting byte(s) will be set to the value of the background color. The foreground and background colors used in the color expansion of monochrome source data

can be set in the source expansion foreground color register and the source expansion background color register.

The BLT Engine requires that the bit alignment of each scan line's worth of monochrome source data be specified. Each scan line's worth of monochrome source data is word aligned but can actually start on any bit boundary of the first byte. Monochrome text is special cased and it is bit or byte packed, where in bit packed there are no invalid pixels (bits) between scan lines. There is a 3 bit field which indicates the starting pixel position within the first byte for each scan line, Mono Source Start.

Note that the Monosource surface start Base Address, should always be Cache Line (64byte) aligned.

The BLT engine also provides various clipping options for use with specific BLT commands (BLT_TEXT) with a monochrome source. Clipping is supported through: Clip rectangle Y addresses or coordinates and X coordinates along with scan line starting and ending addresses (with Y addresses) along with X starting and ending coordinates.

The maximum immediate source size is 128 bytes.

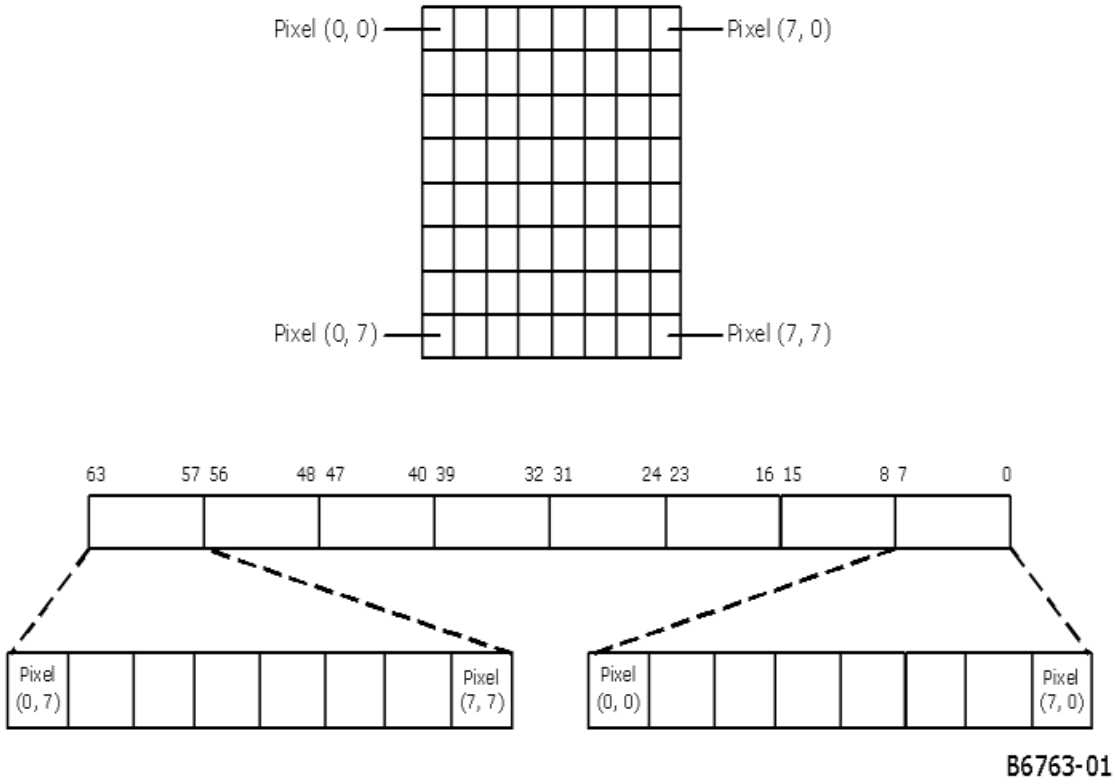
Pattern Data

The color pattern data must exist within the frame buffer or Graphics aperture where the BLT engine may read it directly or it can be sent through the command stream. The pattern data must be located in linear memory. Note also that the Color Pattern surface start Base Address, should always be Cache Line (64byte) aligned.

Monochrome pattern data is supplied by the command packet when it is to be used. As shown in figure below, the block of pattern graphics data always represents a block of 8x8 pixels. The bits or bytes of a block of pattern data may be organized in the frame buffer memory in only one of three ways, depending upon its color depth which may be 8, 16, or 32 bits per pixel (whichever matches the color depth to which the BLT engine has been set), or monochrome.

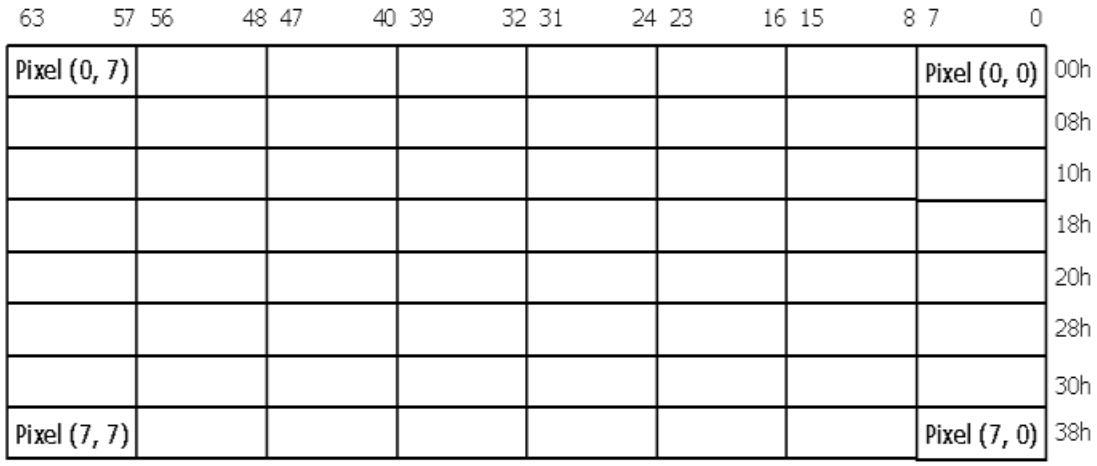
The maximum color pattern size is 256 bytes.

Pattern Data -- Always an 8x8 Array of Pixels



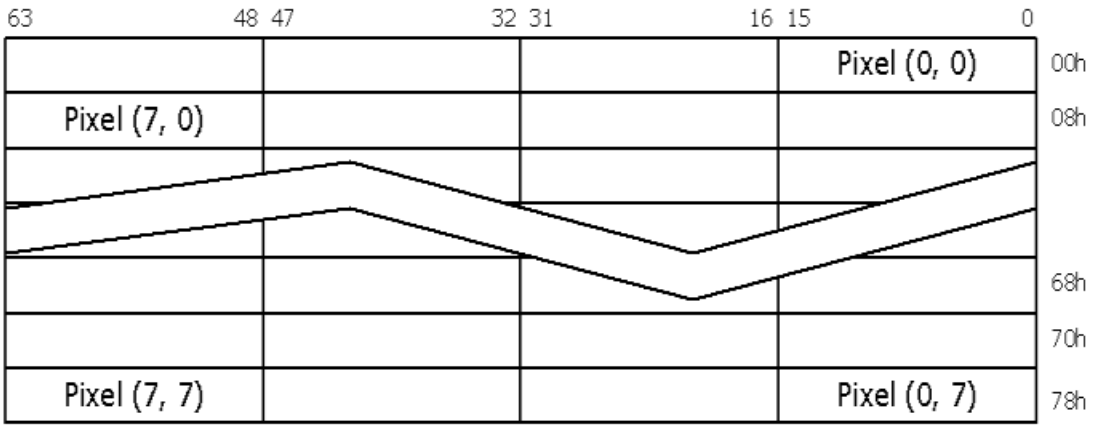
The Pattern Address Register is used to specify the address of the color pattern data at which the block of pattern data begins. The three least significant bits of the address written to this register are ignored, because the address must be in terms of quadwords. This is because the pattern must always be located on an address boundary equal to its size. Monochrome patterns take up 8 bytes, or a single quadword of space, and are loaded through the command packet that uses it. Similarly, color patterns with color depths of 8, 16, and 32 bits per pixel must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The next 3 figures show how monochrome, 8bpp, 16bpp, and 32bpp pattern data , respectively, is organized in memory.

8bpp Pattern Data -- Occupies 64 Bytes (8 quadwords)



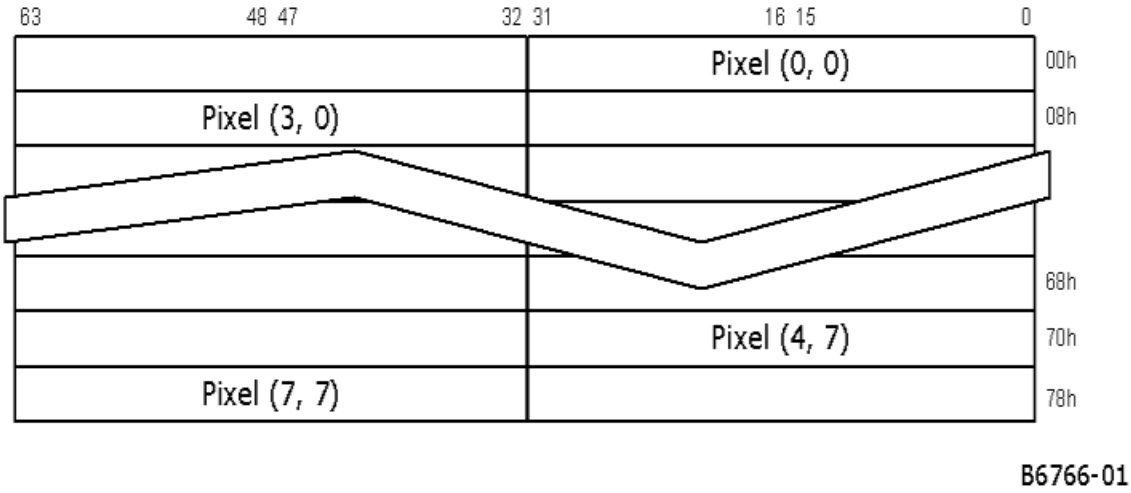
B6764-01

16bpp Pattern Data -- Occupies 128 Bytes (16 quadwords)



B6765-01

32bpp Pattern Data -- Occupies 256 Bytes (32 quadwords)



The opcode of the command packet specifies whether the pattern data is color or monochrome. The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome pattern data, the BLT engine is designed to convert it into color through a process called “color expansion” which takes place as a BLT operation is performed. In color expansion, the single bits of monochrome pattern data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry values corresponding to either the foreground or background color that have been specified for use in this process. The foreground color is used for pixels corresponding to a bit of monochrome pattern data that carry the value of 1, while the background color is used where the corresponding bit of monochrome pattern data carries the value of 0. The foreground and background colors used in the color expansion of monochrome pattern data can be set in the Pattern Expansion Foreground Color Register and Pattern Expansion Background Color Register.

Destination Data

There are actually two different types of “destination data”: the graphics data already residing at the location that is designated as the destination, and the data that is to be written into that very same location as a result of a BLT operation.

The location designated as the destination must be within the frame buffer or Graphics aperture where the BLT engine can read from it and write to it directly. The blocks of destination data to be read from and written to the destination may be either contiguous or discontinuous. All data written to the destination will have the color depth to which the BLT engine has been set. It is presumed that any data already existing at the destination which will be read by the BLT engine will also be of this same color depth — the BLT engine neither reads nor writes monochrome destination data.

The Destination Address Register is used to specify the address of the destination.

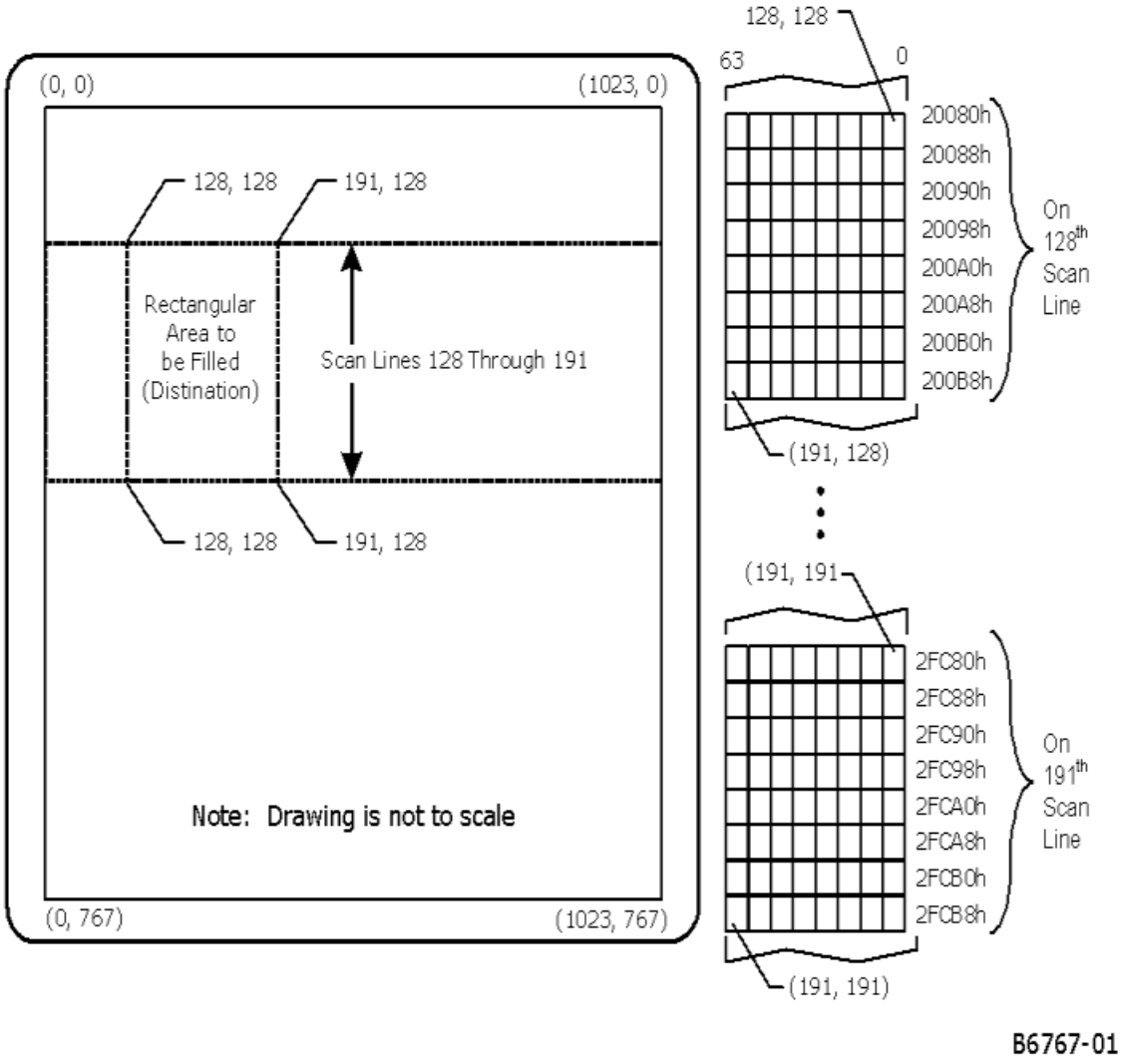
To accommodate discontinuous destination data, the Source and Destination Pitch Registers can be used to specify the offset in bytes from the beginning of one scan line's worth of destination data to the next. Otherwise, if the destination data is contiguous, then an offset equal to the length of a scan line's worth of destination data should be specified.

BLT Programming Examples

Pattern Fill — A Very Simple BLT

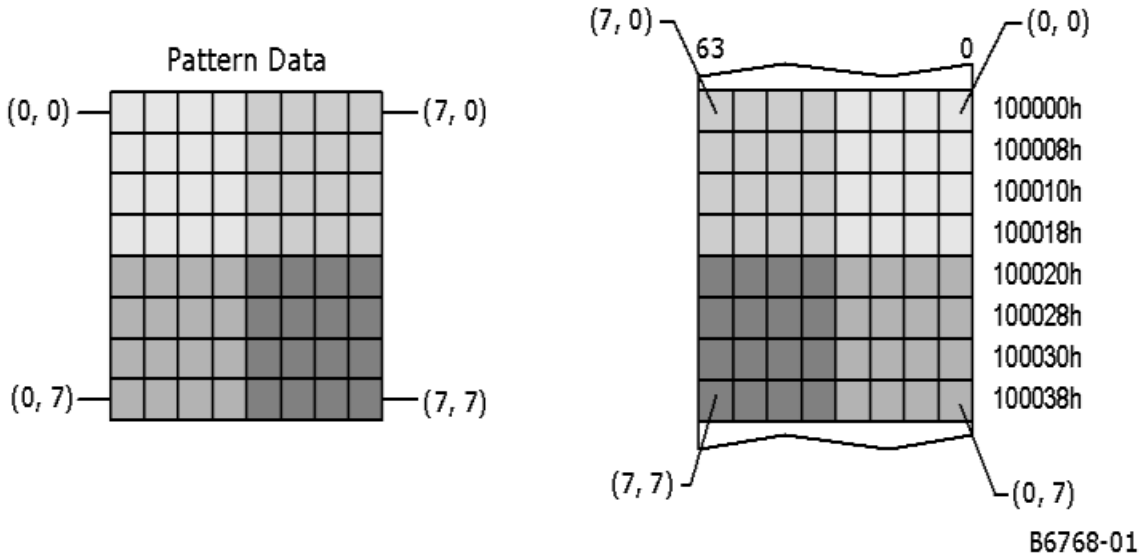
In this example, a rectangular area on the screen is to be filled with a color pattern stored as pattern data in off-screen memory. The screen has a resolution of 1024x768 and the graphics system has been set to a color depth of 8 bits per pixel.

On-Screen Destination for Example Pattern Fill BLT



As shown in the figure above, the rectangular area to be filled has its upper left-hand corner at coordinates (128, 128) and its lower right-hand corner at coordinates (191, 191). These coordinates define a rectangle covering 64 scan lines, each scan line's worth of which is 64 pixels in length — in other words, an array of 64x64 pixels. Presuming that the pixel at coordinates (0, 0) corresponds to the byte at address 00h in the frame buffer memory, the pixel at (128, 128) corresponds to the byte at address 20080h.

Pattern Data for Example Pattern Fill BLT



As shown in figure above, the pattern data occupies 64 bytes starting at address 100000h. As always, the pattern data represents an 8x8 array of pixels.

The BLT command packet is used to select the features to be used in this BLT operation, and must be programmed carefully. The vertical alignment field should be set to 0 to select the top-most horizontal row of the pattern as the starting row used in drawing the pattern starting with the top-most scan line covered by the destination. The pattern data is in color with a color depth of 8 bits per pixel, so the dynamic color enable should be asserted with the dynamic color depth field should be set to 0. Since this BLT operation does not use per-pixel write-masking (destination transparency mode), this field should be set to 0. Finally, the raster operation field should be programmed with the 8-bit value of F0h to select the bit-wise logical operation in which a simple copy of the pattern data to the destination takes place. Selecting this bit-wise operation in which no source data is used as an input causes the BLT engine to automatically forego either reading source data from the frame buffer.

The Destination Pitch Register must be programmed with number of bytes in the interval from the start of one scan line's worth of destination data to the next. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024.

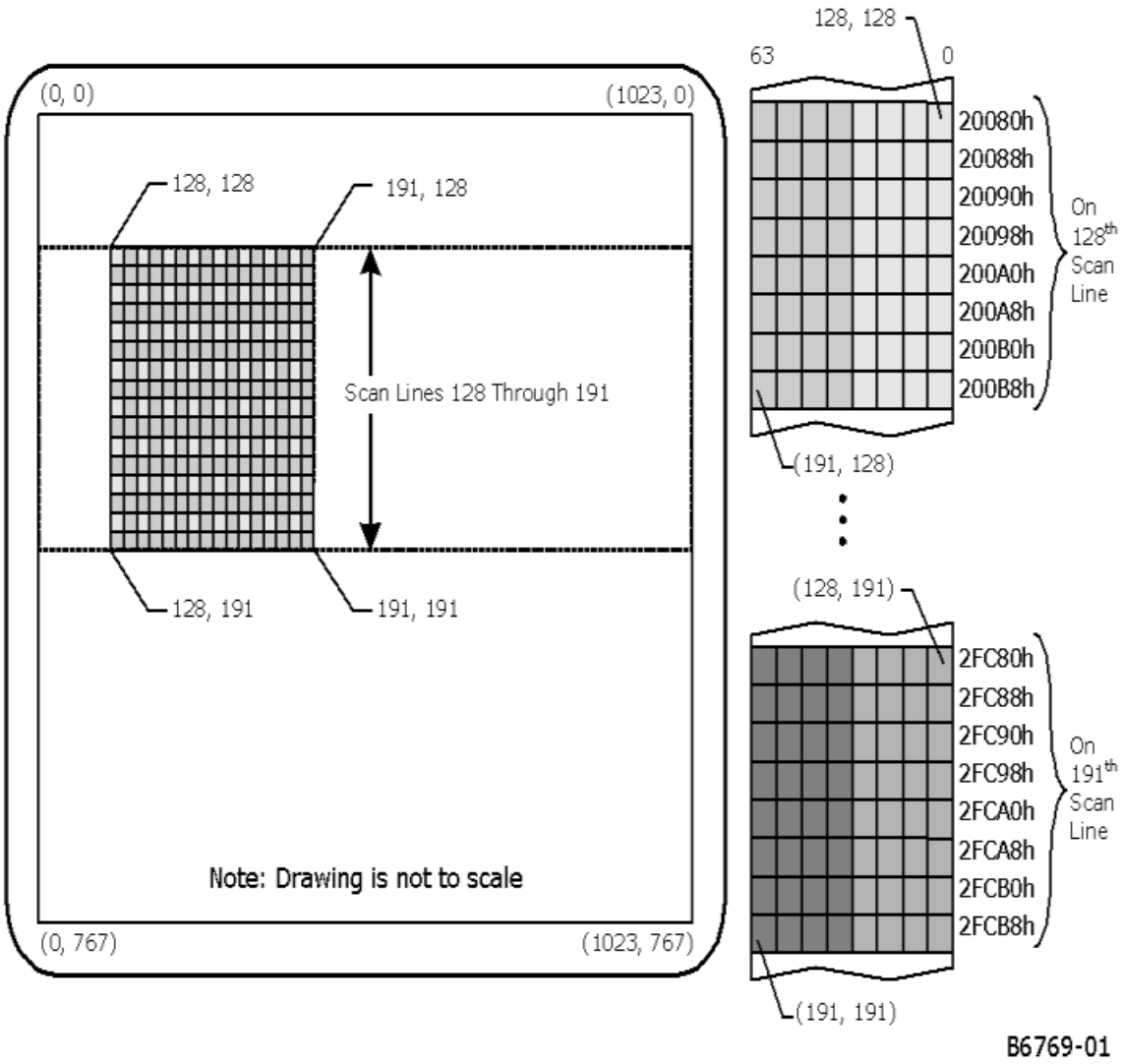
Bits [31:3] of the Pattern Address Register must be programmed with the address of the pattern data.

Similarly, bits [31:0] of the Destination Address Register must be programmed with the byte address at the destination that will be written to first. In this case, the address is 20080h, which corresponds to the byte representing the pixel at coordinates (128, 128).

This BLT operation does not use the values in the Source Address Register or the Source Expansion Background or Foreground Color Registers.

The Destination Width and Height Registers (or the Destination X and Y Coordinates) must be programmed with values that describe to the BLT engine the 64x64 pixel size of the destination location. The height should be set to carry the value of 40h, indicating that the destination location covers 64 scan lines. The width should be set to carry the value of 40h, indicating that each scan line's worth of destination data occupies 64 bytes. All of this information is written to the ring buffer using the PAT_BLT (or XY_PAT_BLT) command packet.

Results of Example Pattern Fill BLT

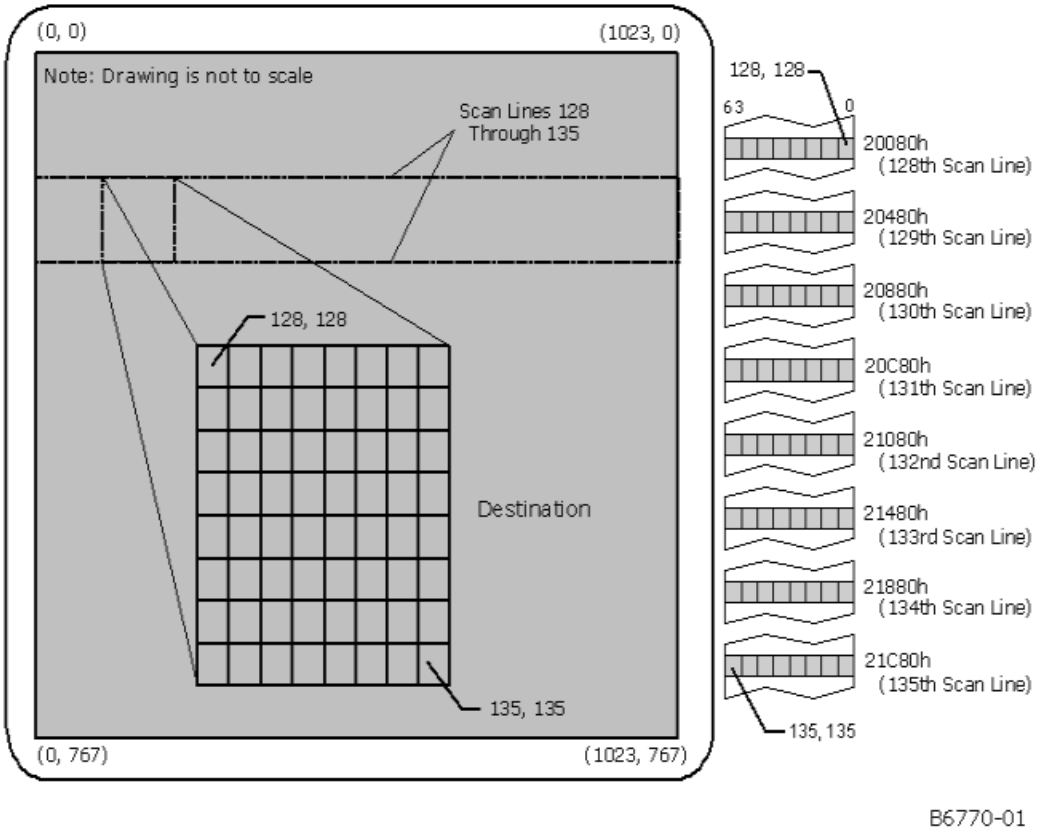


The figure above shows the end result of performing this BLT operation. The 8x8 pattern has been repeatedly copied ("tiled") into the entire 64x64 area at the destination.

Drawing Characters Using a Font Stored in System Memory

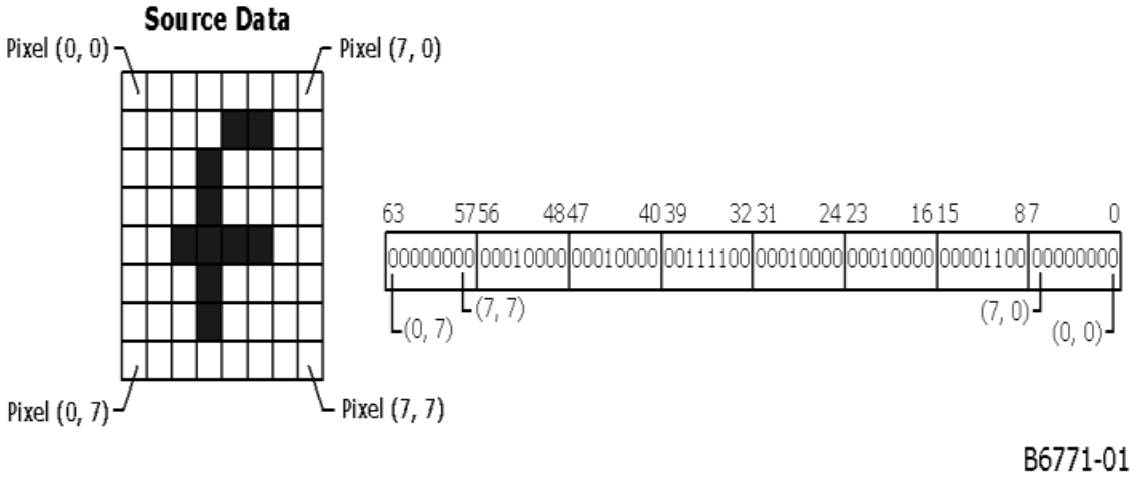
In this example BLT operation, a lowercase letter "f" is to be drawn in black on a display with a gray background. The resolution of the display is 1024x768, and the graphics system has been set to a color depth of 8 bits per pixel.

On-Screen Destination for Example Character Drawing BLT



The figure above shows the display on which this letter "f" is to be drawn. As shown in this figure, the entire display has been filled with a gray color. The letter "f" is to be drawn into an 8x8 region on the display with the upper left-hand corner at the coordinates (128, 128).

Source Data in System Memory for Example Character Drawing BLT



The figure above shows both the 8x8 pattern making up the letter "f" and how it is represented somewhere in the host's system memory — the actual address in system memory is not important. The letter "f" is represented in system memory by a block of monochrome graphics data that occupies 8 bytes. Each byte carries the 8 bits needed to represent the 8 pixels in each scan line's worth of this graphics data. This type of pattern is often used to store character fonts in system memory.

During this BLT operation, the host CPU will read this representation of the letter "f" from system memory, and write it to the BLT engine by performing memory writes to the ring buffer as an immediate monochrome BLT operand following the BLT_TEXT command. The BLT engine will receive this data through the command stream and use it as the source data for this BLT operation. The BLT engine will be set to the same color depth as the graphics system — 8 bits per pixel, in this case. Since the source data in this BLT operation is monochrome, color expansion must be used to convert it to an 8 bpp color depth. To ensure that the gray background behind this letter "f" is preserved, per-pixel write masking will be performed, using the monochrome source data as the pixel mask.

The BLT Setup and Text_immediate command packets are used to select the features to be used in this BLT operation. Only the fields required by these two command packets must be programmed carefully. The BLT engine ignores all other registers and fields. The source select field in the Text_immediate command must be set to 1, to indicate that the source data is provided by the host CPU through the command packet. Finally, the raster operation field should be programmed with the 8-bit value CCh to select the bit-wise logical operation that simply copies the source data to the destination. Selecting this bit-wise operation in which no pattern data is used as an input, causes the BLT engine to automatically forego reading pattern data from the frame buffer.

The Setup Pattern/Source Expansion Foreground Color Register to specify the color with which the letter "f" will be drawn. There is no Source address. All scan lines of the glyph are bit packed and the clipping is controlled by the ClipRect registers from the SETUP_BLT command and the Destination Y1, Y2, X1, and X2 registers in the TEXT_BLT command. Only the pixels that are within (inclusive comparisons) the clip rectangle are written to the destination surface.

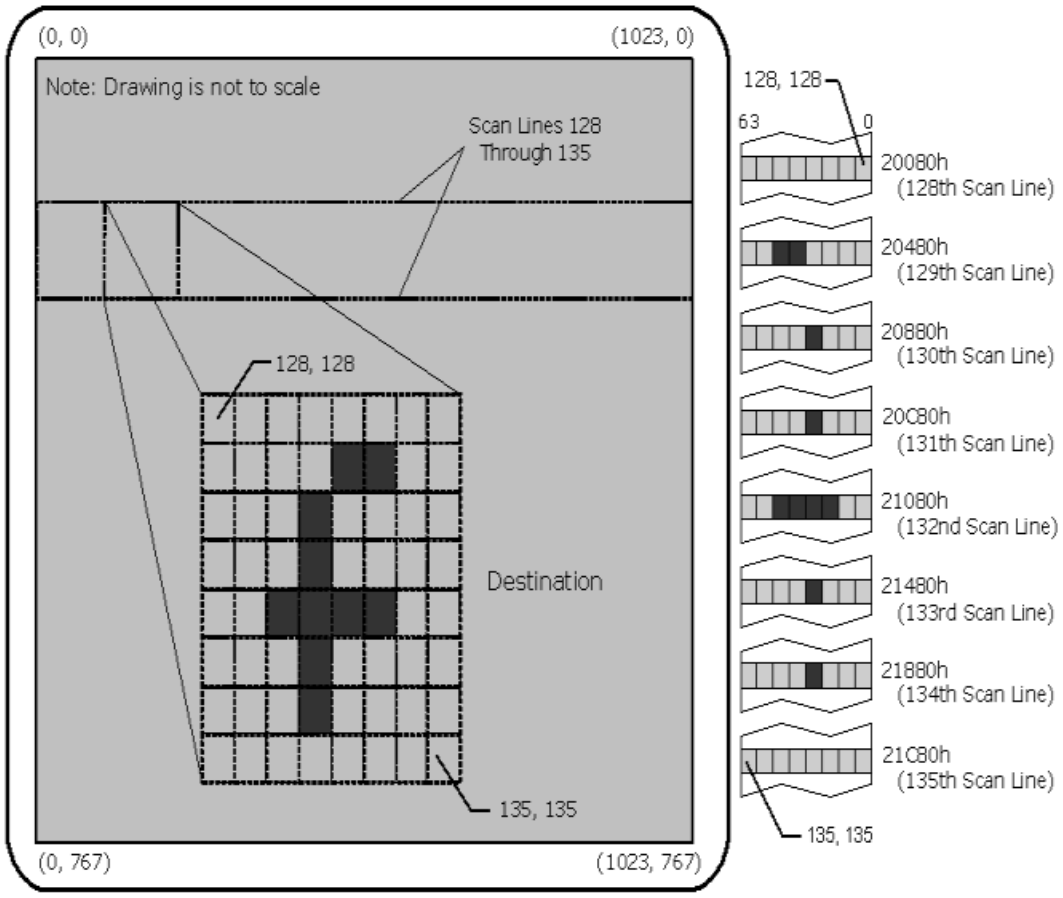
The Destination Pitch Register must be programmed with a value equal to the number of bytes in the interval between the first bytes of each adjacent scan line's worth of destination data. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024 pixels, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024. Since the source data used in this BLT operation is monochrome, the BLT engine will not use a byte-oriented pitch value for the source data.

Since the source data is monochrome, color expansion is required to convert it to color with a color depth of 8 bits per pixel. Since the Setup Pattern/Source Expansion Foreground Color Register is selected to specify the foreground color of black to be used in drawing the letter "f", this register must be programmed with the value for that color. With the graphics system set for a color depth of 8 bits per pixel, the actual colors are specified in the RAMDAC palette, and the 8 bits stored in the frame buffer for each pixel actually specify the index used to select a color from that palette. This example assumes that the color specified at index 00h in the palette is black, and therefore bits [7:0] of this register should be set to 00h to select black as the foreground color. The BLT engine ignores bits [31:8] of this register because the selected color depth is 8 bits per pixel. Even though the color expansion being performed on the source data normally requires that both the foreground and background colors be specified, the value used to specify the background color is not important in this example. Per-pixel write-masking is being performed with the monochrome source data as the pixel mask, which means that none of the pixels in the source data that will be converted to the background color will ever be written to the destination. Since these pixels will never be seen, the value programmed into the Pattern/Source Expansion Background Color Register to specify a background color is not important.

The Destination Width and Height Registers are not used. The Y1, Y2, X1, and X2 are used to describe to the BLT engine the 8x8 pixel size of the destination location. The Destination Y1 and Y2 address (or coordinate) registers must be programmed with the starting and ending scan line address (or Y coordinates) of the destination data. This address is specified as an offset from the start of the frame buffer of the scan line at the destination that will be written to first. The destination X1 and X2 registers must be programmed with the starting and ending pixel offsets from the beginning of the scan line.

This BLT operation does not use the values in the Pattern Address Register, the Source Expansion Background Color Register, or the Source Expansion Foreground Color Register.

Results of Example Character Drawing BLT



B6772-01

The preceding shows the end result of performing this BLT operation. Only the pixels that form part of the actual letter "f" have been drawn into the 8x8 destination location on the display, leaving the other pixels within the destination with their original gray color.

BLT Instruction Overview

This chapter defines the instructions used to control the 2D (BLT) rendering function.

The instructions detailed in this chapter are used across devices. However, slight changes may be present in some instructions (i.e., for features added or removed), or some instructions may be removed entirely. Refer to the *Device Dependencies* chapter for summary information regarding device-specific behaviors/interfaces/features.

The XY instructions offload the drivers by providing X and Y coordinates and taking care of the access directions for overlapping BLTs without fields specified by the driver.

Color pixel sizes supported are 8, 16, and 32 bits per pixel (bpp). All pixels are naturally aligned.

BLT Engine State

Most of the BLT instructions are state-free, which means that all states required to execute the command is within the instruction. If clipping is not used, then there is no shared state for many of the BLT instructions. This allows the BLT Engine to be shared by many drivers with minimal synchronization between the drivers.

Instructions which share state are:

All instructions that are X,Y commands and use the Clipping Rectangle by asserting the Clip Enable field

All XY_Setup Commands (XY_SETUP_BLT and XY_SETUP_MONO_PATTERN_SL_BLT, XY_SETUP_CLIP_BLT) load the shared state for the following commands:

XY_PIXEL_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_SCANLINES_BLT	
XY_TEXT_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_TEXT_IMMEDIATE_BLT	(Negative Stride (=Pitch) Not Allowed)

State registers that are saved & restored in the Logical Context:

BR1+	Setup Control (Solid Pattern Select, Clipping Enable, Mono Source Transparency Mode, Mono Pattern Transparency Mode, Color Depth[1:0], Raster Operation[7:0], & Destination Pitch[15:0]) + 32bpp Channel Mask[1:0], Mono / Color Pattern
BR05	Setup Background Color
BR06	Setup Foreground Color
BR07	Setup Pattern Base Address
BR09	Setup Destination Base Address
BR20	DW0 for a Monochrome Pattern
BR21	DW1 for a Monochrome Pattern
BR24	ClipRectY1’X1

BR25 ClipRectY2’X2

Cacheable Memory Support

The BLT Engine can be used to transfer data between cacheable (“system”) memory and uncached (“main”, or “UC”) graphics memory using the BLT instructions. The GTT must be properly programmed to map memory pages as cacheable or UC. Only linear-mapped (not tiled) surfaces can be mapped as cacheable.

Transfers between cacheable sources and cacheable destinations are not supported. Patterns and monochrome sources can not be located in cacheable memory.

Cacheable write operands do not snoop the processor’s cache nor update memory until evicted from the render cache. Cacheable read or write operands are not snooped (nor invalidated) from either internal cache by external (processor, hublink,…) accesses.

Device Cache Coherency: Render & Texture Caches

Software must initiate cache flushes to enforce coherency between the render and texture caches, i.e., both the render and texture caches must be flushed before a BLT destination surface can be reused as a texture source. Color sources and destinations use the render cache, while patterns and monochrome sources use the texture cache.

BLT Engine Instructions

The Instruction Target field is used as an opcode by the BLT Engine state machine to qualify the control bits that are relevant for executing the instruction. The descriptions for each DWord and bit field are contained in the *BLT Engine Instruction Field Definition* section. Each DWord field is described as a register, but none of these registers can be written or read through a memory mapped location; they are internal state only.

BLT Programming Restrictions

Overlapping Source/Destination BLTs:

For all products *negative pitch* programming is allowed only when the source and destination surfaces are of the same type: linear source to linear destination copy, or tiled source to tiled destination copy. This is a must requirement. In such cases:

- Both the pitches must be programmed to be a negative value, if the source and destination surfaces are overlapping.
- Either of the pitches can be programmed to a negative value if required to do mirroring, but only if the complete source and destination surfaces are not overlapping anywhere on the surfaces.

Description
For the XY_FAST_COPY_BLT instruction, this restriction applies: When two sequential fast copy blits have different source surfaces, but their destinations refer to the same destination surface and therefore destinations overlap, a Flush must be inserted between the two blits.
The **pitch** length for **Linear Surfaces** is OWord-aligned (16-byte multiple), as the BSpec says. (For Tiled surfaces, the pitch length is always Cacheline aligned (64-byte multiple), as Tile surface pitches must be a multiple of Tile widths, which are always cacheline aligned).

Legacy Blits:

The following condition must be avoided when programming the BLT engine: Linear surfaces with a cache line in scan line Y for the source stream overlapping with a cache line in scan line Y-1 for the dest stream (=> non-aligned surface pitches). The cache coherency rules combined with the Blitter data consumption rules result in UNDEFINED operation.

All reserved fields must be programmed to 0s.

When using monosource or text data (bit/byte/word aligned): Do not program pixel widths greater than 32,745 pixels.

The other way to do this is driver should always program a dummy 3D.

NON-PIPELINE state following the BLT commands:

Description
For Monosource and Color Pattern surfaces, and also linear colour source and destination surfaces, the start **Base Address** programmed should always be Cache Line (64 byte) aligned.

Workaround	
Context:	BLT Engine
An XY_FULL_MONOSOURCE_IMMEDIATE_PATTERN command should always be followed by MI_FLUSH_DW.	

Programming Note	
Context:	Fast Copy Blitter
XY_FAST_COPY command with source Linear and destination Tiled Y and height%4 == 3 is not supported. Use the XY_SOURCE_COPY command for such workloads.	

Programming Note	
Context:	Fast Copy Blitter + Frame Buffer Compression
Fast Copy workloads targetting the Front Buffer with FBC enabled is not supported.	

BLT Engine Instruction Field Definitions

This section describes the BLT Engine instruction fields. These descriptions are in the format of register descriptions. These registers are internal and are not readable. Some of these registers are state that is saved and restored for supporting separate software threads.

Register
BR00 - BLT Opcode and Control
BR01 - Setup BLT Raster OP, Control, and Destination Offset
BR05 - Setup Expansion Background Color
BR06 - Setup Expansion Foreground Color
BR07 - Setup Blit Color Pattern Address Lower Order Address bits
BR30 - Setup Blit Color Pattern Address Higher Order Address
BR09 - Destination Address Lower Order Address Bits
BR27 - Destination Address Higher Order Address
BR11 - BLT Source Pitch (Offset)
BR12 - Source Address Lower order Address bits
BR28 - Source Address Higher order Address
BR13 - BLT Raster OP, Control, and Destination Pitch
BR14 - Destination Width and Height
BR15 - Color Pattern Address Lower order Address bits
BR29 - Color Pattern Address Higher order Address
BR16 - Pattern Expansion Background and Solid Pattern Color
BR17 - Pattern Expansion Foreground Color
BR18 - Source Expansion Background and Destination Color
BR19 - Source Expansion Foreground Color

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 12: Display

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

VGA and Extended VGA Registers 1

General Control and Status Registers 2

ST00 - Input Status 0 3

ST01 - Input Status 1 4

FCR - Feature Control 6

MSR - Miscellaneous Output 7

Sequencer Registers 9

SRX - Sequencer Index 9

SR00 - Sequencer Reset 9

SR01 - Clocking Mode 10

SR02 - Plane/Map Mask 11

SR03 - Character Font 12

SR04 - Memory Mode Register 13

SR07 - Horizontal Character Counter Reset 14

Graphics Controller Registers 14

GRX - GRX Graphics Controller Index Register 15

GR00 - Set/Reset Register 15

GR01 - Enable Set/Reset Register 16

GR02 - Color Compare Register 16

GR03 - Data Rotate Register 17

GR04 - Read Plane Select Register 17

GR05 - Graphics Mode Register 18

GR06 - Miscellaneous Register 22

GR07 - Color Don't Care Register 23

GR08 - Bit Mask Register 24

GR10 - Address Mapping 24

GR11 - Page Selector 26

GR18 - Software Flags 26

Attribute Controller Registers 27

ARX - Attribute Controller Index Register 28

AR[00:0F] - Palette Registers [0:F] 28

AR10 - Mode Control Register 29
AR11 - Overscan Color Register 31
AR12 - Memory Plane Enable Register 32
AR13 - Horizontal Pixel Panning Register 33
AR14 - Color Select Register 34
VGA Color Palette Registers 35
DACMASK - Pixel Data Mask Register 36
DACSTATE - DAC State Register 36
DACRX - Palette Read Index Register 37
DACWX - Palette Write Index Register 37
DACDATA - Palette Data Register 38
CRT Controller Register 39
CRX - CRT Controller Index Register 40
CR00 - Horizontal Total Register 40
CR01 - Horizontal Display Enable End Register 41
CR02 - Horizontal Blanking Start Register 41
CR03 - Horizontal Blanking End Register 42
CR04 - Horizontal Sync Start Register 43
CR05 - Horizontal Sync End Register 44
CR06 - Vertical Total Register 45
CR07 - Overflow Register (Vertical) 46
CR08 - Preset Row Scan Register 49
CR09 - Maximum Scan Line Register 50
CR0A - Text Cursor Start Register 51
CR0B - Text Cursor End Register 52
CR0C - Start Address High Register 53
CR0D - Start Address Low Register 53
CR0E - Text Cursor Location High Register 54
CR0F - Text Cursor Location Low Register 54
CR10 - Vertical Sync Start Register 55
CR11 - Vertical Sync End Register 56
CR12 - Vertical Display Enable End Register 57
CR13 - Offset Register 57

CR14 - Underline Location Register 58
CR15 - Vertical Blanking Start Register 59
CR16 - Vertical Blanking End Register 59
CR17 - CRT Mode Control 60
CR18 - Line Compare Register 64
CR22 - Memory Read Latch Data Register 65
CR24 - Toggle State of Attribute Controller Register 65
Display Audio Codec Verbs 66
Block Diagram 66
Codec Node Hierarchy 67
Programming 68
Verb Support 68
Parameter Support 70
Node ID 00h Root Node Verbs 71
F00h - Get Parameters 71
Parameter 00h: VID - Vendor ID 71
Parameter 02h: RID - Revision ID 71
Parameter 04h: PARAM_SNC - Subordinate Node Count 71
F37h GET CCF - Get Current Clock Frequency 72
Node ID 01h Audio Function Group Verbs 72
F00h Get Parameters 72
Parameter 04h: PARAM_SNC - Subordinate Node Count 73
Parameter 05h: PARAM_FGT - Function Group Type 73
Parameter 08h: PARAM_FGC - Function Group Capability 73
Parameter 0Fh: PARAM_SPS - Supported Power States 73
Parameter 16h: PARAM_A2CAP - Azalia 2 Capabilities 74
705h SET_PS - Set Power State 74
F05h GET_PS - Get Power State 74
F20h GET SSID - Get Subsystem ID0 75
720h SET SSID0 - Set Subsystem ID0 75
721h SET SSID1 - Set Subsystem ID1 75
722h SET SSID2 - Set Subsystem ID2 75
723h SET SSID3 - Set Subsystem ID3 75

724h SET CCF - Set Current Clock Frequency....75
F24h GET CCF - Get Current Clock Frequency....76
7FFh SET Function Group Reset....76
Node ID 02h 03h 04h Audio Output Convertor Widget Verbs....76
2hAh SETGET_SDF - SetGET Stream Descriptor Format....77
F00h Get Parameters....77
Parameter 09h: AWC - Audio Widget Capabilities....78
Parameter 0Ah: PSB - PCM Sizes and Bit Rates....79
Parameter 0Bh: SF - Stream Formats....80
Parameter 0Fh: PARAM_SPS - Supported Power States....80
705h SET_PS - Set Power State....80
F05h GET_PS - Get Power State....80
706hF06h GETSET_CSID - GetSet Channel and Stream ID....81
Digital Converter Verbs....81
70Dh: SET_DC1 - Set Digital Converter 1....82
70Eh: Digital Converter 2....82
73Eh: Digital Converter 3....82
73Fh: Digital Converter 4....83
72DhF2Dh GETSET_CCC - GetSet Converter Channel Count....83
Node ID 05h 06h 07h Pin Widget Verbs....83
3h SET_AM - Set Amplifier Mute....84
B8h GET_AM - Get Amplifier Mute....84
F00h Get Parameters....84
Parameter 09h: AWC - Audio Widget Capabilities....85
Parameter 0Ch: PC - Pin Capabilities....85
Parameter 0Eh: CLL - Connection List Length....86
Parameter 12h: OAC - Output Amplifier Capabilities....86
Parameter 15h: DLL - Device List Length....87
Parameter 0Fh: PARAM_SPS - Supported Power States....87
701hF01h SETGET_CSC - SetGet Connection Select Control....87
F02h GET_CLE - Get Connection List Entry....87
705h SET_PS - Set Power State....88
F05h GET_PS - Get Power State....88

707hF07h SETGET_PWC - SetGet Pin Widget Control 88
708hF08h SETGET_UE - SetGet Unsolicited Enable 88
F09h GET_PS - Get Pin Sense 89
71Ch SET_CD0 - Set Configuration Default Byte 0 89
71Dh SET_CD1 - Set Configuration Default Byte 1 89
71Eh SET_CD2 - Set Configuration Default Byte 2 89
71Fh SET_CD3 - Set Configuration Default Byte 3 89
F1Ch GET_CD - Get Configuration Default 90
F2Eh HDMIDP Info Size 90
F2Fh Get ELD Data 91
Parameter nn: ELD Data 91
730hF30h SETGET_HII - SetGet HDMI Info Index 91
731hF31h SETGET_HID - SetGet HDMI Info Data 91
732hF32h SETGET_HITC - SetGet HDMI Info Transmit Control 91
733h SET_PC - Set Protection Control 92
734hF34h SETGET_CCM - GetSet Converter Channel Map 92
735h SET_DS - Set Device Select 92
F35h: GET_DS - Get Device Select 92
F36h GET_DDLE - Get Display Device List Entry 92
73ChF3Ch SETGET_DPID - SetGet DisplayPort Stream ID 93
Node ID 08h Intel Vendor Widget Verbs 93
F00h Get Parameters 94
Parameter 09h: AWC - Audio Widget Capabilities 94
71Eh SET_GET_GFXMAILBOX - Set Get GFX MAILBOX Byte 2 94
728h SET CLOCK OFF - Set Clock Off Command 94
708hF08h SETGET_UE - SetGet Unsolicited Enable 94
781hF81h GETSET_VV - GetSet iDisp Codec Vendor Verb 95
782h SET_GTCT - Set GTC Trigger 95
F83h GET_CGTC - Get Captured GTC Value 95
F84h GET_CWC - Get Captured Wall Clock Value 95
F85h GET GOF - Get GTC Offset Value 96
785h SET GOF0 - Set GTC Offset Value Byte 0 96
786h SET GOF1 - Set GTC Offset Value Byte 1 96

787h SET GOF2 - Set GTC Offset Value Byte 2 96
788h SET GOF3 - GTC Offset Value Byte 3 96
789hF89h SETGET_GDI - SetGet GTC Device Index 96
3h SET_AM - Set Amplifier Mute 96
B8h GET_AM - Get Amplifier Mute 97
F00h Get Parameters 97
Parameter 09h: AWC - Audio Widget Capabilities 97
Parameter 0Ch: PC - Pin Capabilities 98
Parameter 0Eh: CLL - Connection List Length 99
Parameter 12h: OAC - Output Amplifier Capabilities 99
Parameter 0Fh: PARAM_SPS - Supported Power States 99
701hF01h SETGET_CSC - SetGet Connection Select Control 100
F02h GET_CLE - Get Connection List Entry 100
705h SET_PS - Set Power State 100
F05h GET_PS - Get Power State 100
707hF07h SETGET_PWC - SetGet Pin Widget Control 101
708hF08h SETGET_UE - SetGet Unsolicited Enable 101
F09h GET_PS - Get Pin Sense 101
71Ch SET_CD0 - Set Configuration Default Byte 0 102
71Dh SET_CD1 - Set Configuration Default Byte 1 102
71Eh SET_CD2 - Set Configuration Default Byte 2 102
71Fh SET_CD3 - Set Configuration Default Byte 3 102
F1Ch GET_CD - Get Configuration Default 103
F2Fh Get ELD Data 103
Parameter nn: ELD Data 103
733h SET_PC - Set Protection Control 103
734hF34h SETGET_CCM - GetSet Converter Channel Map 104
740hF40h SETGET PTS Offset Byte0 104
741hF41h SETGET PTS Offset Byte1 104
742hF42h SETGET PTS Offset Byte2 104
743hF43h SETGET PTS Offset Byte3 104
North Display Engine Registers 105
Display Connections 105

Display Pipes 106
Display Transcoders 107
Audio 107
DDIs 107
Pipe to Transcoder to DDI Mappings 109
Terminology 110
Mode Set 112
Sequences to Initialize Display 112
Initialize Sequence 112
Un-initialize Sequence 113
Sequences for DisplayPort 113
Enable Sequence 113
Notes 114
Enabling DisplayPort Sync Mode 115
Disable Sequence 115
Disabling DisplayPort Sync Mode 116
Sequences for HDMI and DVI 117
Enable Sequence 117
Notes 117
Disable Sequence 118
Sequences for Display C5 and C6 119
Sequence to Allow DC5 or DC6 119
Sequence to Disallow DC5 and DC6 120
DMC Firmware Package 120
CSS Header 122
Package Header 122
DMC firmware binary 123
Display Resolution Support 124
Maximum Pipe Pixel Rate 124
Maximum Port Link Rate 125
Maximum Memory Read Bandwidth 125
Maximum Watermark 126
Display Resolution Capabilities 126

Examples 127
Clocks 128
Overview of Supported Display Clock Paths 129
Display Engine Clock Reference 130
Display Engine PLLs 130
Recommended PLL Selection 131
DDI Clocks 131
Transcoder Clocks 131
CD Clock 132
Port Clock Programming 133
DisplayPort Programming 133
DisplayPort PLL Enable Sequence 133
DisplayPort PLL Disable Sequence 133
Example of DisplayPort on DDIA using HBR 2.7 GHz link rate with SSC 133
HDMI and DVI Programming 134
HDMI and DVI PLL Enable Sequence 134
HDMI and DVI PLL Disable Sequence 134
Formula for HDMI and DVI DPLL Programming 134
Algorithm to Find HDMI and DVI DPLL Programming 134
Pseudo-code for HDMI and DVI DPLL Programming 135
Example of DVI on DDIB using 113.309 MHz symbol clock 136
Example of HDMI on DDIC using 296.703 MHz symbol clock 137
Sequences for Changing CD Clock Frequency 138
Resets 139
Shared Functions 140
Fuses and Straps 140
Interrupts 140
Interrupt Flow 141
Interrupt Service Routine 142
Render Response 143
Arbiter 143
GSA 143
Data Buffer 143

Backlight 144
Backlight Enabling Sequence 144
Backlight Registers 145
Miscellaneous Shared Functions 145
Central Power 145
Frame Buffer Compression 145
FBC Registers 145
FBC Overview 145
FBC Compression Limit 146
FBC Programming Overview 146
Render Tracking With Nuke 147
Render Tracking Without Nuke 147
Blitter Tracking With Nuke 148
Blitter Tracking Without Nuke 148
CPU Host Aperture Tracking 149
Display Plane Enabling with FBC 149
Display Plane Disabling with FBC 149
Watermarks 150
DC States 150
Power Wells 150
Pipe 150
Color Space Conversion 150
Pipe Color Gamut Enhancement 153
Pipe DPST 155
Pipe Palette and Gamma 155
Programming Modes 155
8 bit legacy palette/gamma mode: 155
10 bit gamma mode: 156
Split gamma mode: 156
12 bit interpolated gamma mode: 157
Example Pipe Gamma Correction Curve 158
Pipe Control 159
Pipe Scaler 159

Planes 160
Plane Capability and Interoperability 160
Plane Assignments and Capabilities 160
Plane Feature Interoperability 161
Render/Display Decompression 162
Color Control Surface 162
Decompression Programming 163
Plane Rotation Programming 163
90 Rotation 163
270 rotation 167
Display Buffer Programming 167
Display Buffer Allocation 167
Display Buffer Size 167
Allocation Requirements 167
Minimum Allocation Requirements 168
Basic Allocation Method 168
Single Pipe 168
Multi-Pipe 169
Buffer allocation re-distribution 170
Display Buffer Allocation and Watermark programming prior to OS boot 170
VGA 171
Cursor Plane 171
Universal Plane 171
Plane Pixel Formats 173
Transcoder 174
Transcoder Control 174
Transcoder Timing 174
Transcoder MN Values 174
Transcoder Video Data Island Packet 176
Transcoder DDI Function 179
Panel Self Refresh 179
Transcoder Port Sync 180
Feature Description 180

DP/eDP Port Sync Restrictions 180
Audio 180
Audio Bios Programming Sequence 180
Codec Verb Table 180
Audio Programming Sequence for Link wakeup 183
Audio Programming Sequence 184
Audio Configuration 186
Digital Display Interface 186
DDI Buffer 186
I_boost 187
Recommended Buffer Translation Programming 187
DDI AUX Channel 190
AUX programming sequence 191
DisplayPort Transport 192
Global Time Code (GTC) 192
Top Level GTC 192
DDI Level GTC 192
South Display Engine Registers 193
Terminology 194
Shared Functions 195
Fuses and Straps 195
Raw Clock 195
Interrupts and Hot Plug 195
Panel Power and Backlight 196
Panel Power 196
Backlight 196
Backlight Enabling Sequence 197
Backlight Registers 197
GMBUS and GPIO 198
Registers 198
Pin Usage 198
GPIO Programming for I2C Bit Bashing 199
GMBUS Controller Programming Interface 199

Sequence for GMBUS Burst Reads Greater Than 511 Bytes 199
Display Watermark Programming 200
Watermark Overview 200
Watermark Calculations 200
Watermark Algorithm 201
Transition Watermark 204
Scaling 205
System Agent Geyserville (SAGV) 206
Sequence to Disable SAGV 206
Sequence to Enable SAGV 206
Examples 207
Example pixel rate adjustments: 207
Example method, block, and line calculations: 207
Memory Values 208
Retrieve Memory Latency Data 208
Memory Latency Data Definition 208
SAGV Block Time 209

VGA and Extended VGA Registers

This section describes the registers and the functional operation notations for the observable registers in the VGA section. This functionality is provided as a means for support of legacy applications and operating systems. It is important to note that these registers in general have the desired effects only when running VGA display modes.

The main exceptions to this are the palette interface which allows real mode DOS applications and full screen VGA applications under an OS control running in high resolution (non-VGA) modes to access the palette through the VGA register mechanisms and the use of the ST01 status bits that determine when the VGA enters display enable and sync periods. Other exceptions include the register bits that control the memory accesses through the A000:0000 and B000:0000 memory segments which are used during operating system emulation of VGA for "DOS box" applications. Some of the functions of the VGA are enabled or defeated through the programming of the VGA control register bits that are located in the MMIO register space.

Given the legacy nature of this function, it has been adapted to the changing environment that it must operate within. The three most notable changes are the addition of high resolution display mode support, new operating system support, and the use of fixed resolution display devices (such as LCD panels). Additional control bits in the PCI Config space will affect the ability to access the registers and memory aperture associated with VGA.

Mode of Operation	VGA Disable	VGA Display	VGA Registers	Palette (VGA)	VGA Memory	VGA Banking
VGA DOS	No	Yes	Yes	Yes	Yes	No
HiRes DOS	Yes	No	Yes	Yes	No	Yes
Fullscreen DOS	Yes/No	No/Yes	Yes	Yes	Yes	Yes
DOS Emulation	Yes	No	Yes	Yes	Yes	Yes

VGA Display Mode	Dot Clock Select	Dot Clock Range	132 Column Text Support	9-Dot Disable Support	Main Use
Native	VGA Clock Select	25/28 MHz	No	No	Analog CRT (VGA connector)
Centered	Fixed at display Requirements	Product Specific	No	Yes	Digital Display
Upper Left Corner	Fixed at display Requirements	Product Specific	No	Yes	Internal Panel

Native, Centered, and Upper Left Corner support varies from product to product.

Even in the native VGA display operational modes, not all combinations of bit settings result in functional operating modes. VGA display modes have the restriction that they can be used only when all other display planes are disabled.

These registers are accessed via I/O space. The I/O space resides in the PCI compatibility hole and uses only the addresses that were part of the original VGA I/O space (which includes EGA and MDA emulation). Accesses to the VGA I/O addresses are steered to the proper bus and rely on proper setup of bridge registers. Extended VGA registers such as GR10 and GR11 use additional indexes for the already defined I/O addresses. VGA register accesses are allowed as 8 or 16 bit naturally aligned transactions only. Word transactions must have the least significant bit of the address set to zero. DWORD I/O operations should not be performed on these registers.

Some products may support access to these registers through MMIO. The access method varies and is documented elsewhere.

General Control and Status Registers

The setup, enable, and general registers are all directly accessible by the CPU. A sub indexing scheme is not used to read from and write to these registers.

Name	Function	Read		Write	
		I/O	Memory Offset	I/O	Memory Offset
ST00	VGA Input Status Register 0	3C2h	3C2h	--	--
ST01	VGA Input Status Register 1	3BAh/3DAh[1]	3BAh/3DAh[1]	--	--
FCR	VGA Feature Control Register	3CAh	3CAh	3BAh/3DAh[1]	3BAh/3DAh[1]
MSR	VGA Miscellaneous Output Register	3CCh	3CCh	3C2h	3C2h

[1] The address selection for ST01 reads and FCR writes is dependent on CGA or MDA emulation mode as selected via the MSR register.

Various bits in these registers provide control over the real-time status of the horizontal sync signal, the horizontal retrace interval, the vertical sync signal, and the vertical retrace interval. The horizontal retrace interval is the period during the drawing of each scan line containing active video data, when the active video data is not being displayed. This period includes the horizontal front and back porches, and the horizontal sync pulse. The horizontal retrace interval is always longer than the horizontal sync pulse. The vertical retrace interval is the period during which the scan lines not containing active video data are drawn. This includes the vertical front porch, back porch, and the vertical sync pulse. The vertical retrace interval is normally longer than the vertical sync pulse.

ST00 - Input Status 0

I/O (and Memory Offset) Address: 3C2h

Default: 00h

Attributes: Read Only

Bit	Descriptions
7	**CRT Interrupt Pending.** This bit is here for EGA compatibility and **will always return zero**. The generation of interrupts was originally enabled, through bits [4,5] of the Vertical Retrace End Register (CR11). This ability to generate interrupts at the start of the vertical retrace interval is a feature that is typically unused by DOS software and therefore is only supported through other means for use under a operating system support. 0 = CRT (vertical retrace interval) interrupt is not pending. 1 = CRT (vertical retrace interval) interrupt is pending
6:5	**Reserved.** Read as 0s.
4	**RGB Comparator / Sense.** This bit is here for compatibility and **will always return one**. Monitor detection must be done be done through the programming of registers in the MMIO space. 0 = Below threshold 1 = Above threshold
3:0	**Reserved.** Read as 0s.

ST01 - Input Status 1

I/O (and Memory Offset) Address: 3BAh/3DAh

Default: 00h

Attributes: Read Only

The address selection is dependent on CGA or MDA emulation mode as selected via the MSR register.

Bit	Descriptions
7	**Reserved (as per VGA specification).** Read as 0s.
6	**Reserved.** Read as 0.
5:4	**Video Feedback 1, 0.** These bits are connected to 2 of the 8 color bits sent to the palette. Bits 4 and 5 of the Color Plane Enable Register (AR12) selects which two of the 8 possible color bits become connected to these 2 bits of this register. These bits exist for EGA compatibility.
3	**Vertical Retrace/Video.** 0 = VSYNC inactive (Indicates that a vertical retrace interval is not taking place). 1 = VSYNC active (Indicates that a vertical retrace interval is taking place). VGA pixel generation is not locked to the display output but is loosely coupled. A VSYNC indication may not occur during the actual VSYNC going to the display but during the VSYNC that is generated as part of the VGA pixel generation. The exact relationship will vary with the VGA display operational mode. This status bit will remain active when the VGA is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now incorrectly) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. Bits 4 and 5 of the Vertical Retrace End Register (CR11) previously could program this bit to generate an interrupt at the start of the vertical retrace interval. This ability to generate interrupts at the start of the vertical retrace interval is a feature that is largely unused by legacy software. Interrupts are not supported through the VGA register bits.
2:1	**Reserved.** Read as 0s.

Bit	Descriptions
0	**Display Enable Output.** Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. This bit was used with the EGA graphics system (and the ones that preceded it, including MDA and CGA). In those cases, it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to the frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer. Those behaviors resulted in either "snow" or a flickering display. This bit provides compatibility with software designed for those early graphics controllers. This bit is currently used in DOS applications that access the palette to prevent the sparkle associated with read and write accesses to the palette RAM with the same address on the same clock cycle. **This status bit remains active when the VGA display is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now considered incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. When in panel fitting VGA or centered VGA operation, the meaning of these bits will not be consistent with native VGA timings.** 0 = Active display data is being sent to the display. Neither a horizontal retrace interval or a vertical retrace interval is currently taking place. 1 = Either a horizontal retrace interval (horizontal blanking) or a vertical retrace interval (vertical blanking) is currently taking place.

FCR - Feature Control

I/O (and Memory Offset) Address: 3BAh/3DAh - Write; 3CAh - Read

Default: 00h

Attributes: See Address above

The I/O address used for writes is dependent on CGA or MDA emulation mode as selected via the MSR register. In the original EGA, bits 0 and 1 were used as part of the feature connector interface. Feature connector is not supported in these devices and those bits will always read as zero.

Bit	Descriptions
7:4	**Reserved.** Read as 0.
3	**VSYNC Control.** This bit is provided for compatibility only and has no other function. Reads and writes to this bit have no effect other than to change the value of this bit. The previous definition of this bit selected the output on the VSYNC pin. 0 = Was used to set VSYNC output on the VSYNC pin (default). 1 = Was used to set the logical 'OR' of VSYNC and Display Enable output on the VSYNC pin. This capability was not typically very useful.
2:0	**Reserved.** Read as 0.

MSR - Miscellaneous Output

I/O (and Memory Offset) Address: 3C2h - Write; 3CCh - Read

Default: 00h

Attributes: See Address above

Bit	Descriptions
7	**CRT VSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. Sync polarity was used in VGA to signal the monitor how many lines of active display are being generated. 0 = Positive Polarity (default). 1 = Negative Polarity.
6	**CRT HSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. 0 = Positive Polarity (default). 1 = Negative Polarity
5	**Page Select.** In Odd/Even Memory Map Mode 1 (GR6), this bit selects the upper or lower 64 KB page in display memory for CPU access: 0 = Upper page (default) 1 = Lower page. Selects between two 64KB pages of frame buffer memory during standard VGA odd/even modes (modes 0h through 5h). Bit 1 of register GR06 can also program this bit in other modes. This bit is would normally set to 1 by the software.
4	**Reserved.** Read as 0.
3:2	**Clock Select.** These bits can select the dot clock source for the CRT interface. The bits should be used to select the dot clock in standard native VGA modes only. When in the centering or upper left corner modes, these bits should be set to have no effect on the clock rate. The actual frequencies that these bits select, if they have any affect at all, is programmable through the DPLL MMIO registers. 00 = CLK0, 25.175 MHz (for standard VGA modes with 640 pixel (8-dot) horizontal resolution) (default) 01 = CLK1, 28.322 MHz. (for standard VGA modes with 720 pixel (9-dot) horizontal resolution) 10 = Was used to select an external clock (now unused) 11 = Reserved

Bit	Descriptions
1	**A0000-BFFFFh Memory Access Enable.** VGA Compatibility bit enables access to video memory (frame buffer) at A0000-BFFFFh. When disabled, accesses to VGA memory are blocked in this region. This bit is independent of and does not block CPU access to the video linear frame buffer at other addresses. 0 = Prevent CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture (default). 1 = Allow CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture. This memory must be mapped as UC by the CPU.
0	**I/O Address Select.** This bit selects 3Bxh or 3Dxh as the I/O address for the CRT Controller registers, the Feature Control Register (FCR), and Input Status Register 1 (ST01). Presently ignored (whole range is claimed), but will "ignore" 3Bx for color configuration or 3Dx for monochrome. It is typical in AGP chipsets to shadow this bit and properly steer I/O cycles to the proper bus for operation where a MDA exists on another bus such as ISA. 0 = Select 3Bxh I/O address (MDA emulation) (default). 1 = Select 3Dxh I/O address (CGA emulation).

In standard VGA modes using the analog VGA connector, bits 7 and 6 indicate which of the three standard VGA vertical resolutions the standard VGA display should use. Extended modes, including those with a vertical resolution of 480 scan lines, may use a setting of 0 for both of these bits. Different connector standards and timing standards specify the proper use of sync polarity. This setting was "reserved" in the VGA standard.

Analog CRT Display Sync Polarities

V	H	Display	Horizontal Frequency	Vertical Frequency
P	P	200 Line	15.7 KHz	60 Hz
N	P	350 Line	21.8 KHz	60 Hz
P	N	400 Line	31.5 KHz	70 Hz
N	N	480 Line	31.5 KHz	60 Hz

Sequencer Registers

The sequencer registers are accessed via either I/O space or Memory space. To access registers the VGA Sequencer Index register (SRX) at I/O address 3C4h (or memory address 3C4h) is written with the index of the desired register. Then the desired register is accessed through the data port for the sequencer registers at I/O address 3C5 (or memory address 3C5).

SRX - Sequencer Index

I/O (and Memory Offset) Address: 3C4h

Default: 00h

Attributes: Read/Write

Bit	Description
7:3	**Reserved.** Read as 0s.
2:0	**Sequencer Index.** This field contains a 3-bit Sequencer Index value used to access sequencer data registers at indices 0 through 7.

SR00 - Sequencer Reset

I/O (and Memory Offset) Address: 3C5h(Index=00h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:2	**Reserved.** Read as 0.
1	**Reserved.** Reserved for VGA compatibility (was reset).
0	**Reserved.** Reserved for VGA compatibility. (was reset)

SR01 - Clocking Mode

I/O (and Memory Offset) Address: 3C5h (Index=01h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
5	**Screen Off.** 0 = Normal Operation (default). 1 = Disables video output (blanks the screen) and turns off display data fetches. Synchronization pulses to the display, however, are maintained. Setting this bit to 1 had been used as a way to more rapidly update and improve CPU access performance to the frame buffer during VGA modes. In non-VGA modes (VGA Disable=1), this bit has no effect. Before the VGA is disabled through the MMIO VGA control register, this bit should be set to stop the memory accesses from the display. The following sequence must be used when disabling the VGA plane. 1. Write SR01 to set bit 5 = 1 to disable video output. 2. Wait for 100us. 3. Disable the VGA plane via Bit 31 of the MMIO VGA control register (location found in the MMIO display register programming specification).
4	**Shift 4.** 0 = Load video shift registers every 1 or 2 character clocks (depending on bit 2 of this register) (default). 1 = Load shift registers every 4th character clock.
3	**Dot Clock Divide.** Setting this bit to 1 stretches doubles all horizontal timing periods that are specified in the VGA horizontal CRTC registers. This bit is used in standard VGA 40-column text modes to stretch timings to create horizontal resolutions of either 320 or 360 pixels (as opposed to 640 or 720 pixels, normally used in standard VGA 80-column text modes). The effect of this is that there will actually be twice the number of pixels sent to the display per line. 0 = Pixel clock is left unaltered (used for 640 (720) pixel modes); (default). 1 = Pixel clock divided by 2 (used for 320 (360) pixel modes).
2	**Shift Load.** Bit 4 of this register must be 0 for this bit to be effective. 0 = Load video data shift registers every character clock (default). 1 = Load video data shift registers every other character clock.
1	**Reserved.** Read as 0.

Bit	Descriptions
0	**8/9 Dot Clocks.** This bit determines whether a character clock is 8 or 9 dot clocks long if clock doubling is disabled and 16 or 18 clocks if it is. This also changes the interpretation of the pixel panning values (see chart). An additional control bit determines if this bit is to be ignored and 8-dot characters are to be used always. The 9-dot disable would be used when doubling the horizontal pixels on a 1280 wide display or non-doubling on a 640 wide display. Panning however will occur according to the expected outcome. 0 = 9 dot clocks (9 horizontal pixels) per character in text modes with a horizontal resolution of 720 pixels. 1 = 8 dot clocks (8 horizontal pixels) per character in text or graphics modes with a horizontal resolution of 640 pixels.

SR02 - Plane/Map Mask

I/O (and Memory Offset) Address: 3C5h (Index=02h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:4	**Reserved.** Read as 0s.
3:0	**Memory Planes [3:0] Processor Write Access Enable.** In both the Odd/Even Mode and the Chain 4 Mode, these bits still control access to the corresponding color plane. 0 = Disable. 1 = Enable. This register is referred to in the VGA standard as the Map Mask Register.

SR03 - Character Font

I/O (and Memory Offset) Address: 3C5h (index=03h)

Default: 00h

Attributes: Read/Write

In text modes, bit 3 of the video data's attribute byte normally controls the foreground intensity. This bit may be redefined to control switching between character sets. This latter function is enabled whenever there is a difference in the values of the Character Font Select A and the Character Font Select B bits. If the two values are the same, the character select function is disabled and attribute bit 3 controls the foreground intensity.

Bit 1 of the Memory Mode Register (SR04) must be set to 1 for the character font select function of this register to be active. Otherwise, only character maps 0 and 4 are available.

<table>
<tr><th>Bit</th><th>Descriptions</th></tr>
<tr><td>7:6</td><td>Reserved. Read as 0s.</td></tr>
<tr><td>3:2,5</td><td>Character Map Select Bits for Character Map B. These three bits are used to select the character map (character generator tables) to be used as the secondary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [3:2,5]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
<tr><td>10,0</td><td>2</td><td>5th 8KB of plane 2 at offset 32K</td></tr>
<tr><td>10,1</td><td>6</td><td>6th 8KB of plane 2 at offset 40K</td></tr>
<tr><td>11,0</td><td>3</td><td>7th 8KB of plane 2 at offset 48K</td></tr>
<tr><td>11,1</td><td>7</td><td>8th 8KB of plane 2 at offset 56K</td></tr>
</table></td></tr>
<tr><td>1:0,4</td><td>Character Map Select Bits for Character Map A. These three bits are used to select the character map (character generator tables) to be used as the primary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [1:0,4]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
<tr><td>10,0</td><td>2</td><td>5th 8KB of plane 2 at offset 32K</td></tr>
<tr><td>10,1</td><td>6</td><td>6th 8KB of plane 2 at offset 40K</td></tr>
<tr><td>11,0</td><td>3</td><td>7th 8KB of plane 2 at offset 48K</td></tr>
<tr><td>11,1</td><td>7</td><td>8th 8KB of plane 2 at offset 56K</td></tr>
</table></td></tr>
</table>

SR04 - Memory Mode Register

I/O (and Memory Offset) Address: 3C5h (index=04h)

Default: 00h

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3	**Chain 4 Mode.** The selections made by this bit affect both CPU Read and write accesses to the frame buffer. 0 = The manner in which the frame buffer memory is mapped is determined by the setting of bit 2 of this register (default). 1 = The frame buffer memory is mapped in such a way that the function of address bits 0 and 1 are altered so that they select planes 0 through 3. This setting is used in mode x13 to allow all four planes to be accessed via sequential addresses.
2	**Odd/Even Mode.** Bit 3 of this register must be set to 0 for this bit to be effective. The selections made by this bit affect only non-paged CPU accesses to the frame buffer through the VGA aperture. 0 = The frame buffer memory is mapped in such a way that the function of address bit 0 such that even addresses select planes 0 and 2 and odd addresses select planes 1 and 3 (default). 1 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).
1	**Extended Memory Enable.** This bit must be set to 1 to enable the selection and use of character maps in plane 2 via the Character Map Select Register (SR03). 0 = Disable CPU accesses to more than the first 64KB of VGA standard memory (default). 1 = Enable CPU accesses to the rest of the 256KB total VGA memory beyond the first 64KB.
0	**Reserved.** Read as 0.

SR07 - Horizontal Character Counter Reset

I/O (and Memory Offset) Address: 3C5h (index=07h)

Default: 00h

Attributes: Read/Write

For standard VGAs, writing this register (with any data) causes the horizontal character counter to be held in reset (the character counter output will remain 0). It remained in reset until a write occurred to any other sequencer register location with SRX set to an index of 0 through 6. In this implementation that sequence has no such special effect.

The vertical line counter is clocked by a signal derived from the horizontal display enable (which does not occur if the horizontal counter is held in reset). Therefore, if a write occurs to this register during the vertical retrace interval, both the horizontal and vertical counters will be set to 0. A write to any other sequencer register location (with SRX set to an index of 0 through 6) may then be used to start both counters with reasonable synchronization to an external event via software control. Although this was a standard VGA register, it was not documented.

Bit	Description
7:0	**Horizontal Character Counter.**

Graphics Controller Registers

The graphics controller registers are accessed via either I/O space or Memory space. Accesses to the registers of the VGA Graphics Controller are done through the use of address 3CEh (or memory address 3CEh) written with the index of the desired register. Then the desired register is accessed through the data port for the graphics controller registers at I/O address 3CFh (or memory address 3CFh). Indexes 10 and 11 should only be accessed through the I/O space only.

GRX - GRX Graphics Controller Index Register

I/O (and Memory Offset) Address: 3CEh

Default: 000UUUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:0	**Graphics Controller Register Index.** This field selects any one of the graphics controller registers (GR00-GR18) to be accessed via the data port at I/O (or memory offset) location 3CFh.

GR00 - Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (index=00h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Set/Reset Plane [3:0].** When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 0, all 8 bits of each byte of each memory plane are set to either 1 or 0 as specified in the corresponding bit in this register, if the corresponding bit in the Enable Set/Reset Register (GR01) is set to 1. When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 3, all CPU data written to the frame buffer is rotated, then logically ANDed with the contents of the Bit Mask Register (GR08), and then treated as the addressed data's bit mask, while value of these four bits of this register are treated as the color value.

GR01 - Enable Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (Index=01h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Enable Set/Reset Plane [3:0].** This register works in conjunction with the Set/Reset Register (GR00). The Write Mode bits (bits 0 and 1) must be set for Write Mode 0 for this register to have any effect. 0 = The corresponding memory plane can be read from or written to by the CPU without any special bitwise operations taking place. 1 = The corresponding memory plane is set to 0 or 1 as specified in the Set/Reset Register (GR00).

GR02 - Color Compare Register

I/O (and Memory Offset) Address: 3CFh (Index=02h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Color Compare Plane [3:0].** When the Read Mode bit (bit 3) of the Graphics Mode Register (GR05) is set to select Read Mode 1, all 8 bits of each byte of each of the 4 memory planes of the frame buffer corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison, wherein value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.

GR03 - Data Rotate Register

I/O (and Memory Offset) Address: 3CFh (Index=03h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:3	**Function Select.** These bits specify the logical function (if any) to be performed on data that is meant to be written to the frame buffer (using the contents of the memory read latch) just before it is actually stored in the frame buffer at the intended address location. 00 = Data being written to the frame buffer remains unchanged, and is simply stored in the frame buffer. 01 = Data being written to the frame buffer is logically ANDed with the data in the memory read latch before it is actually stored in the frame buffer. 10 = Data being written to the frame buffer is logically ORed with the data in the memory read latch before it is actually stored in the frame buffer. 11 = Data being written to the frame buffer is logically XORed with the data in the memory read latch before it is actually stored in the frame buffer.
2:0	**Rotate Count.** These bits specify the number of bits to the right to rotate any data that is meant to be written to the frame buffer just before it is actually stored in the frame buffer at the intended address location.

GR04 - Read Plane Select Register

I/O (and Memory Offset) Address: 3CFh (Index=04h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**Read Plane Select.** These two bits select the memory plane from which the CPU reads data in Read Mode 0. In Odd/Even Mode, bit 0 of this register is ignored. In Chain 4 Mode, both bits 1 and 0 of this register are ignored. The four memory planes are selected as follows: 00 = Plane 0

Bit	Description
	01 = Plane 1 10 = Plane 2 11 = Plane 3 These two bits also select which of the four memory read latches may be read via the Memory read Latch Data Register (CR22). The choice of memory read latch corresponds to the choice of plane specified in the table above. The Memory Read Latch Data register and this additional function served by 2 bits are features of the VGA standard that were never documented.

GR05 - Graphics Mode Register

I/O (and Memory Offset) Address: 3CFh (Index=05h)

Default: 0UUU U0UUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Shift Register Control.** In standard VGA modes, pixel data is transferred from the 4 graphics memory planes to the palette via a set of 4 serial output bits. These 2 bits of this register control the format in which data in the 4 memory planes is serialized for these transfers to the palette. **Bits [6:5]=00** One bit of data at a time from parallel bytes in each of the 4 memory planes is transferred to the palette via the 4 serial output bits, with 1 of each of the serial output bits corresponding to a memory plane. This provides a 4-bit value on each transfer for 1 pixel, making possible a choice of 1 of 16 colors per pixel. Serial Out / 1st Xfer / 2nd Xfer / 3rd Xfer / 4th Xfer / 5th Xfer / 6th Xfer / 7th Xfer / 8th Xfer Bit 3 / plane3 bit7 / plane3 bit6 / plane3 bit5 / plane3 bit4 / plane3 bit3 / plane3 bit2 / plane3 bit1 / plane3 bit0 Bit 2 / plane2 bit7 / plane2 bit6 / plane2 bit5 / plane2 bit4 / plane2 bit3 / plane2 bit2 / plane2 bit1 / plane2 bit0 Bit 1 / plane1 bit7 / plane1 bit6 / plane1 bit5 / plane1 bit4 / plane1 bit3 / plane1 bit2 / plane1 bit1 / plane1 bit0 Bit 0 / plane0 bit7 / plane0 bit6 / plane0 bit5 / plane0 bit4 / plane0 bit3 / plane0 bit2 / plane0 bit1 / plane0 bit0

Bit	Description
	Bits [6:5]=01 Two bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that alternates per byte between memory planes 0 and 2, and memory planes 1 and 3. First the even-numbered and odd-numbered bits of a byte in memory plane 0 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of a byte in memory plane 2 are transferred via serial output bits 2 and 3. Next, the even-numbered and odd-numbered bits of a byte in memory plane 1 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of memory plane 3 are transferred via serial out bits 1 and 3. This provides a pair of 2-bit values (one 2-bit value for each of 2 pixels) on each transfer, making possible a choice of 1 of 4 colors per pixel. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane2 bit7 plane2 bit5 plane2 bit3 plane2 bit1 plane3 bi7t plane3 bit5 plane3 bit3 plane3 bit1 Bit 2 plane2 bit6 plane2 bit4 plane2 bit2 plane2 bit0 plane3 bit6 plane3 bit4 plane3 bit2 plane3 bit0 Bit 1 plane0 bit7 plane0 bit5 plane0 bit3 plane0 bit1 plane1 bit7 plane1 bit5 plane1 bit3 plane1 bit1 Bit 0 plane0 bit6 plane0 bit4 plane0 bit2 plane0 bit0 plane1 bit6 plane1 bit4 plane1 bit2 plane1 bit0 This alternating pattern is meant to accommodate the use of the Odd/Even mode of organizing the 4 memory planes, which is used by standard VGA modes 2h and 3h. **Bits [6:5]=1x** Four bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that iterates per byte through memory planes 0 through 3. First the 4 most significant bits of a byte in memory plane 0 are transferred via the 4 serial output bits, followed by the 4 least significant bits of the same byte. Next, the same transfers occur from the parallel byte in memory planes 1, 2 and lastly, 3. Each transfer provides either the upper or lower half of an 8 bit value for the color for each pixel, making possible a choice of 1 of 256 colors per pixel. This is the setting used in mode x13. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane0 plane0 plane1 plane1 plane2 plane2 plane3 plane3

Bit	Description
	<table><tr><td></td><td>bit7</td><td>bit3</td><td>bit7</td><td>bit3</td><td>bit7</td><td>bit3</td><td>bit7</td><td>b3it</td></tr><tr><td>Bit 2</td><td>plane0 bit6</td><td>plane0 bit2</td><td>plane1 bit6</td><td>plane1 bit2</td><td>plane2 bit6</td><td>plane2 bit2</td><td>plane3 bit6</td><td>plane3 bit2</td></tr><tr><td>Bit 1</td><td>plane0 bit5</td><td>plane0 bit1</td><td>plane1 bit5</td><td>plane1 bit1</td><td>plane2 bit5</td><td>plane2 bit1</td><td>plane3 bit5</td><td>plane3 bit1</td></tr><tr><td>Bit 0</td><td>plane0 bit4</td><td>plane0 bit0</td><td>plane1 bit4</td><td>plane1 bit0</td><td>plane2 bit4</td><td>plane2 bit0</td><td>plane3 bit4</td><td>plane3 bit0</td></tr></table> This pattern is meant to accommodate mode 13h, a standard VGA 256-color graphics mode.
4	**Odd/Even Mode.** 0 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02). 1 = The frame buffer is mapped in such a way that the function of address bit 0 is such that even addresses select memory planes 0 and 2 and odd addresses select memory planes 1 and 3. This works in a way that is the inverse of (and is normally set to be the opposite of) bit 2 of the Memory Mode Register (SR02).
3	**Read Mode.** 0 = During a CPU read from the frame buffer, the value returned to the CPU is the memory plane selected by bits 1 and 0 of the Read Plane Select Regist 1 = During a CPU read from the frame buffer, all 8 bits of the byte in each of the 4 memory planes corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison. A value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all 4 of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.
2	**Reserved.** Read as 0.

Bit	Description
1:0	**Write Mode.** 00 = Write Mode 0 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written with the CPU write data after it has been rotated by the number of counts specified in the Data Rotate Register (GR03). If, however, the bit(s) in the Enable Set/Reset Register (GR01) corresponding to one or more of the memory planes is set to 1, then those memory planes will be written to with the data stored in the corresponding bits in the Set/Reset Register (GR00). 01 = Write Mode 1 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written to with the data stored in the memory read latches. (The memory read latches stores an unaltered copy of the data last read from any location in the frame buffer.) 10 = Write Mode 2 - During a CPU write to the frame buffer, the least significant 4 data bits of the CPU write data is treated as the color value for the pixels in the addressed byte in all 4 memory planes. The 8 bits of the Bit Mask Register (GR08) are used to selectively enable or disable the ability to write to the corresponding bit in each of the 4 memory planes that correspond to a given pixel. A setting of 0 in a bit in the Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with value of their counterparts in the memory read latches. A setting of 1 in a Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with the 4 bits taken from the CPU write data to thereby cause the pixel corresponding to these bits to be set to the color value. 11 = Write Mode 3 - During a CPU write to the frame buffer, the CPU write data is logically ANDed with the contents of the Bit Mask Register (GR08). The result of this ANDing is treated as the bit mask used in writing the contents of the Set/Reset Register (GR00) are written to addressed byte in all 4 memory planes.

GR06 - Miscellaneous Register

I/O (and Memory Offset) Address: 3CFh (Index=06h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0s.
3:2	**Memory Map Mode.** These 2 bits control the mapping of the VGA address range for frame buffer into the CPU address space as follows: 00 = A0000h - BFFFFh 01 = A0000h - AFFFFh 10 = B0000h - B7FFFh 11 = B8000h - BFFFFh This function is used in standard VGA modes, extended VGA modes (132 column text), and in non-VGA modes (hi-res). 132 column text modes are no longer supported. VGA aperture memory accesses are also controlled by the PCI configuration Memory Enable bit and MSR<1>. For accesses using GR10 and GR11 to paged VGA RAM or to device MMIO registers, set these bits to 01 to select the (A0000-AFFFF) range. The CPU must map this memory as uncacheable (UC).
1	**Chain Odd/Even.** This bit provides the ability to alter the interpretation of address bit A0, so that it may be used in selecting between the odd-numbered memory planes (planes 1 and 3) and the even-numbered memory planes (planes 0 and 2). 0 = A0 functions normally. 1 = A0 is switched with a high order address bit, in terms of how it is used in address decoding. The result is that A0 is used to determine which memory plane is being accessed (A0=0 for planes 0 and 2 and A0=1 for planes 1 and 3).

Bit	Description
0	**Graphics/Text Mode.** This is one of two bits that are used to determine if the VGA is operating in text or graphics modes. The other bit is in AR10[0], these two bits need to be programmed in a consistent manner to achieve the proper results. 0 = Text mode. 1 = Graphics mode.

GR07 - Color Don't Care Register

I/O (and Memory Offset) Address: 3CFh (Index=07h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Ignore Color Plane [3:0].** These bits have effect only when bit 3 of the Graphics Mode Register (GR05) is set to 1 to select read mode 1. 0 = The corresponding bit in the Color Compare Register (GR02) will not be included in color comparisons. 1 = The corresponding bit in the Color Compare Register (GR02) is used in color comparisons.

GR08 - Bit Mask Register

I/O (and Memory Offset) Address: 3CFh (Index=08h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Bit Mask.** 0 = The corresponding bit in each of the 4 memory planes is written to with the corresponding bit in the memory read latches. 1 = Manipulation of the corresponding bit in each of the 4 memory planes via other mechanisms is enabled. This bit mask applies to any writes to the addressed byte of any or all of the 4 memory planes, simultaneously. This bit mask is applicable to any data written into the frame buffer by the CPU, including data that is also subject to rotation, logical functions (AND, OR, XOR), and Set/Reset. To perform a proper read-modify-write cycle into frame buffer, each byte must first be read from the frame buffer by the CPU (and this will cause it to be stored in the memory read latches), this Bit Mask Register must be set, and the new data then written into the frame buffer by the CPU.

GR10 - Address Mapping

I/O (avoid MMIO access) Address: 3CFh (Index=10h)

Default: 00h

Attributes: R/W

This register should only be accessed using I/O operations and never be accessed through the A/B segment addressing map, I/O space register map, or direct MMIO operations.

Bit	Description
7:4	**Page Select Extension - Unused** These bits form the upper bits of a 12-bit page selection value. When combined with the GR11 <7:0> bits they define the offset into stolen memory to the 64KB page that is accessible via the VGA Memory paging mechanism. These bits are ignored.
3	**Reserved**

Bit	Description
2:1	**Paging Map Target.** When paging is enabled, these bits determine the target for data cycle accesses through the VGA memory aperture. VGA graphics memory starts from the base of graphics data stolen memory defined in the PCI configuration BDSM register. VGA display uses the first four 64KB pages of VGA graphics memory. 00 = VGA Graphics Memory 01 = Reserved 10 = Reserved 11 = Reserved
0	**Page Mapping Enable.** This mode allows the mapping of the VGA memory address space. Once this is enabled, no VGA memory address swizzle will be performed, addresses are directly mapped to memory. A single paging register is used to map the 64KB [A0000:AFFFF] window. An internal address is generated using GR11 as the address lines extension to the lower address lines of the access A[15:2]. When mapping is enabled, the B0000:BFFFF area must be disabled using GR06<3:2>=01. The use of addresses in the A0000-BFFFF range require that both the graphics device PCI configuration memory enable and MSR<1> be enabled. 0 = Disable (default) 1 = Enable

GR11 - Page Selector

I/O (avoid MMIO access) Address: 3CFh (Index=11h)

Default: 00h

Attributes: R/W

Bit	Description
7	**Reserved**
6:0	**Page Select.** When concatenated with the GR10<7:4> bits, selects a 64KB window within target area when Page Mapping is enabled (GR10[0]=1). This requires that the graphics device PCI configuration space memory enable, the GR06<3:2> bits to be 01 (select A0000-AFFFF only), and the MSR<1:1> bit to be set. This register provides the Address[22:16] bits for the access. VGA paging of frame buffer memory is for non-VGA packed modes only and should not be enabled when using basic VGA modes. This register should only be accessed using I/O operations.

GR18 - Software Flags

I/O (and Memory Offset) Address: 3CFh (Index=18h)

Default: 00

Attributes: R/W

Bit	Description
7:0	**Software Flags.** Used as scratch pad space in video BIOS. These bits are separate from the bits which appear in the memory mapped IO space. They are used specifically by the SMI BIOS which does not have access to memory mapped IO at the time they are required. These register bits have no effect on H/W operation.

Attribute Controller Registers

Unlike the other sets of indexed registers, the attribute controller registers are not accessed through a scheme employing entirely separate index and data ports. I/O address 3C0h (or memory address 3C0h) is used both as the read and write for the index register, and as the write address for the data port. I/O address 3C1h (or memory address 3C1h) is the read address for the data port.

To write to the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is written to the very same I/O (memory) address. A flip-flop alternates with each write to I/O address 3C0h (or memory address 3C0h) to change its function from writing the index to writing the actual data, and back again. This flip-flop may be deliberately set so that I/O address 3C0h (or memory address 3C0h) is set to write to the index (which provides a way to set it to a known state) by performing a read operation from Input Status Register 1 (ST01) at I/O address 3BAh (or memory address 3BAh) or 3DAh (or memory address 3DAh), depending on whether the graphics system has been set to emulate an MDA or a CGA as per MSR[0].

To read from the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is read from I/O address 3C1h (or memory address 3C1h). A read operation from I/O address 3C1h (or memory address 3C1h) does not reset the flip-flop to writing to the index. Only a write to 3C0h (or memory address 3C0h) or a read from 3BAh or 3DAh (or memory address 3BAh or 3DAh), as described above, will toggle the flip-flop back to writing to the index.

ARX - Attribute Controller Index Register

I/O (and Memory Offset) Address: 3C0h

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0s.
5	**Video Enable.** In the VGA standard, this is called the "Palette Address Source" bit. Clearing this bit will cause the VGA display data to become all 00 index values. For the default palette, this will cause a black screen. The video timing signals continue. Another control bit will turn video off and stop the data fetches. 0 = Disable. Attribute controller color registers (AR[00:0F]) can be accessed by the CPU. 1 = Enable. Attribute controller color registers (AR[00:0F]) are inaccessible by the CPU.
4:0	**Attribute Controller Register Index.** These five bits are used to select any one of the attribute controller registers (AR[00:14]), to be accessed.

AR[00:0F] - Palette Registers [0:F]

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=00h-0Fh)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:0	**Palette Bits P[5:0].** In each of these 16 registers, these are the lower 6 of 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors available to be selected in the palette. Bits 3 and 2 of the Color Select Register (AR14) supply bits P7 and P6 for the values contained in all 16 of these registers. Bits 1 and 0 of the Color Select Register (AR14) can also replace bits P5 and P4 for the values contained in all 16 of these registers, if bit 7 of the Mode Control Register (AR10) is set to 1.

AR10 - Mode Control Register

I/O (and Memory Offset) Address:	Read at 3C1h and Write at 3C0h; (index=10h)
Default:	UUh (U=Undefined)
Attributes:	Read/Write

Bit	Description
7	**Palette Bits P5, P4 Select.** **0** = P5 and P4 for each of the 16 selected colors (for modes that use 16 colors) are individually provided by bits 5 and 4 of their corresponding Palette Registers (AR[00:0F]). **1** = P5 and P4 for all 16 of the selected colors (for modes that use 16 colors) are provided by bits 1 and 0 of Color Select Register (AR14).
6	**Pixel Width/Clock Select.** **0** = Six bits of video data (translated from 4 bits via the palette) are output every dot clock. **1** = Two sets of 4 bits of data are assembled to generate 8 bits of video data which is output every other dot clock, and the Palette Registers (AR[00:0F]) are bypassed. This bit is set to 0 for all of the standard VGA modes, except mode 13h.
5	**Pixel Panning Compatibility.** **0** = Scroll both the upper and lower screen regions horizontally as specified in the Pixel Panning Register (AR13). **1** = Scroll only the upper screen region horizontally as specified in the Pixel Panning Register (AR13). This bit has application only when split-screen mode is being used, where the display area is divided into distinct upper and lower regions which function somewhat like separate displays.
4	**Reserved.** Read as 0.
3	**Enable Blinking/Select Background Intensity.** **0** = Disables blinking in graphics modes, and for text modes, sets bit 7 of the character attribute bytes to control background intensity, instead of blinking. **1** = Enables blinking in graphics modes and for text modes, sets bit 7 of the character attribute bytes to control blinking, instead of background intensity. The blinking rate is derived by dividing the VSYNC signal. The Blink Rate Control field of the VGA control register defines the blinking rate.

Bit	Description
2	**Enable Line Graphics Character Code.** **0** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the background of the character of which the given pixel is a part. **1** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the 8th pixel if the character of which the given pixel is a part. This setting is intended to accommodate the line-drawing characters of the PC's extended ASCII character set -- characters with an extended ASCII code in the range of B0h to DFh. In some literature describing the VGA standard, the range of extended ASCII codes that are said to include the line-drawing characters is mistakenly specified as C0h to DFh, rather than the correct range of B0h to DFh.
1	**Select Display Type.** **0** = Attribute bytes in text modes are interpreted as they would be for a color display. **1** = Attribute bytes in text modes are interpreted as they would be for a monochrome display.
0	**Graphics/Alphanumeric Mode.** This bit (along with GR06[0]) select either graphics mode or text mode. These two bits must be programmed in a consistent manner to achieve the desired results. **0** = Alphanumeric (text) mode. **1** = Graphics mode.

AR11 - Overscan Color Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=11h)

Default: UUh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:0	**Overscan.** These 8 bits select the overscan (border) color index value. The actual border color will be determined by the contents of the palette at the selected index. The border color is displayed between the end of active and the beginning of blank or the end of blank and the beginning of active on CRT type devices driven from the DAC output port. For native VGA modes on digital display ports there is the option of including the border in the active region or not depending on a control bit in the port control register. For centered VGA modes, the VGA control register determines if the border is included in the centered region or not. For monochrome displays, this value should be set to 00h.

AR12 - Memory Plane Enable Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=12h)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:4	**Video Status Mux.** These 2 bits are used to select 2 of the 8 possible palette bits (P7-P0) to be made available to be read via bits 5 and 4 of the Input Status Register 1 (ST01). The table below shows the possible choices. Bit [5:4] / ST01 Bit 5 / ST01 Bit 4 00 / P2 (default) / P0 (default) 01 / P5 / P4 10 / P3 / P1 11 / P7 / P6 These bits are typically unused by current software; they are provided for EGA compatibility.
3:0	**Enable Plane [3:0].** These 4 bits individually enable the use of each of the 4 memory planes in providing 1 of the 4 bits used in video output to select 1 of 16 possible colors from the palette to be displayed. 0 = Disable the use of the corresponding memory plane in video output to select colors, forcing the bit that the corresponding memory plane would have provided to a value of 0. 1 = Enable the use of the corresponding memory plane in video output to select colors. AR12 is referred to in the VGA standard as the Color Plane Enable Register.

AR13 - Horizontal Pixel Panning Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=13h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:0	**Horizontal Pixel Shift 3-0.** This field holds a 4-bit value that selects the number of pixels by which the image is shifted horizontally to the left. This function is available in both text and graphics modes and allows for pixel panning. In text modes with a 9-pixel wide character box, the image can be shifted up to 9 pixels to the left. In text modes with an 8-pixel wide character box, and in graphics modes other than those with 256 colors, the image can be shifted up to 8 pixels to the left. A pseudo 9-bit mode is when the 9-dot character is selected but overridden by the VGA control bit. In standard VGA mode 13h (where bit 6 of the Mode Control Register, AR10, is set to 1 to support 256 colors), bit 0 of this register must remain set to 0, and the image may be shifted up to only 4 pixels to the left. In this mode, the number of pixels by which the image is shifted can be further controlled using bits 6 and 5 of the Preset Row Scan Register (CR08).

Number of Pixels Shifted				
Bits [3:0]	**9-dot**	**Pseudo 9-dot**	**8-dot**	**256-Color**
0	1	1	0	0
1	2	2	1	Undefined
2	3	3	2	1
3	4	4	3	Undefined
4	5	5	4	2
5	6	6	5	Undefined
6	7	7	6	3
7	8	7	7	Undefined
8	0	0	Undefined	Undefined

AR14 - Color Select Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=14h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:2	**Palette Bits P[7:6].** These are the 2 upper-most of the 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors contained in the palette. These 2 bits are common to all 16 sets of bits P5 through P0 that are individually supplied by Palette Registers 0-F (AR[00:0F]).
1:0	**Alternate Palette Bits P[5:4].** These 2 bits can be used as an alternate version of palette bits P5 and P4. Unlike the P5 and P4 bits that are individually supplied by Palette Registers 0-F (AR[00:0F]), these 2 alternate palette bits are common to all 16 of Palette Registers. Bit 7 of the Mode Control Register (AR10) is used to select between the use of either the P5 and P4 bits that are individually supplied by the 16 Palette Registers or these 2 alternate palette bits.

VGA Color Palette Registers

In devices that have multiple display pipes, there is one palette for each display pipe. These palettes are the same for VGA modes and non-VGA modes. Accesses through VGA register methods can optionally read or write from either one.

For each palette, the color data stored in these 256 color data positions can be accessed only through a complex sub-addressing scheme, using a data register and two index registers. The Palette Data Register at I/O address 3C9h (or memory address offset 3C1h) is the data port. The Palette Read Index Register at I/O address 3C7h (or memory address offset 3C7h) and the Palette Write Index Register at I/O address 3C8h (or memory address offset 3C8h) are the two index registers. The Palette Read Index Register is the index register that is used to choose the color data position that is to be read from via the data port, while the Palette Write Index Register is the index register that is used to choose the color data position that is to be written to through the same data port. This arrangement allows the same data port to be used for reading from and writing to two different color data positions. Reading and writing the color data at a color data position involves three successive reads or writes since the color data stored at each color data position consists of three bytes.

To read a palette color data position, the index of the desired color data position must first be written to the Palette Read Index Register. Then all three bytes of data in a given color data position may be read at the Palette Data Register. The first byte read from the Palette Data Register retrieves the 8-bit value specifying the intensity of the red color component. The second and third bytes read are the corresponding 8-bit values for the green and blue color components respectively. After completing the third read operation, the Palette Read Index Register is automatically incremented so that the data of the next color data position becomes accessible for being read. This allows the contents of all of the 256 color data positions of the palette to be read in sequence. This is done by specifying only the index of the 0th color data position in the Palette Read Index Register, and then simply performing 768 successive reads from the Palette Data Register.

Writing a color data position, entails a very similar procedure. The index of the desired color data position must first be written to the Palette Write Index Register. Then all three bytes of data to specify a given color may be written to the Palette Data Register. The first byte written to the Palette Data Register specifies the intensity of the red color component, the second byte specifies the intensity for the green color component, and the third byte specifies the same for the blue color component. One important detail is that all three of these bytes must be written before the hardware will actually update these three values in the given color data position. When all three bytes have been written, the Palette Write Index Register is automatically incremented so that the data of the next color data position becomes accessible for being written. This allows the contents of all of the 256 color data positions of the palette to be written in sequence. This is done by specifying only the index of the 0th color data position in the Palette Write Index Register, and then simply performing 768 successive writes to the Palette Data Register.

DACMASK - Pixel Data Mask Register

I/O (and Memory Offset) Address: 3C6h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Pixel Data Mask.** In indexed-color mode, the 8 bits of this register are logically ANDed with the 8 bits of pixel data received from the frame buffer for each pixel. The result of this ANDing process becomes the actual index used to select color data positions within the palette. This has the effect of limiting the choice of color data positions that may be specified by the incoming 8-bit data. 0 = Corresponding bit in the resulting 8-bit index being forced to 0. 1 = Allows the corresponding bit in the resulting index to reflect the actual value of the corresponding bit in the incoming 8-bit pixel data.

DACSTATE - DAC State Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Read Only

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**DACState.** This field indicates which of the two index registers was most recently written. **Bits [1:0]** Index Register Indicated 00 = Palette Write Index Register at I/O Address 3C7h (default) 01 = Reserved 10 = Reserved 11 = Palette Read Index Register at I/O Address 3C8h

DACRX - Palette Read Index Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Read Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being read from via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been read. A write to this register will abort a uncompleted palette write sequence. This register allows access to the palette even when running non-VGA display modes.

DACWX - Palette Write Index Register

I/O (and Memory Offset) Address: 3C8h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Write Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being written via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been written. This register allows access to the palette even when running non-VGA display modes.

DACDATA - Palette Data Register

I/O (and Memory Offset) Address: 3C9h

Default: Undefined

Attributes: Read/Write

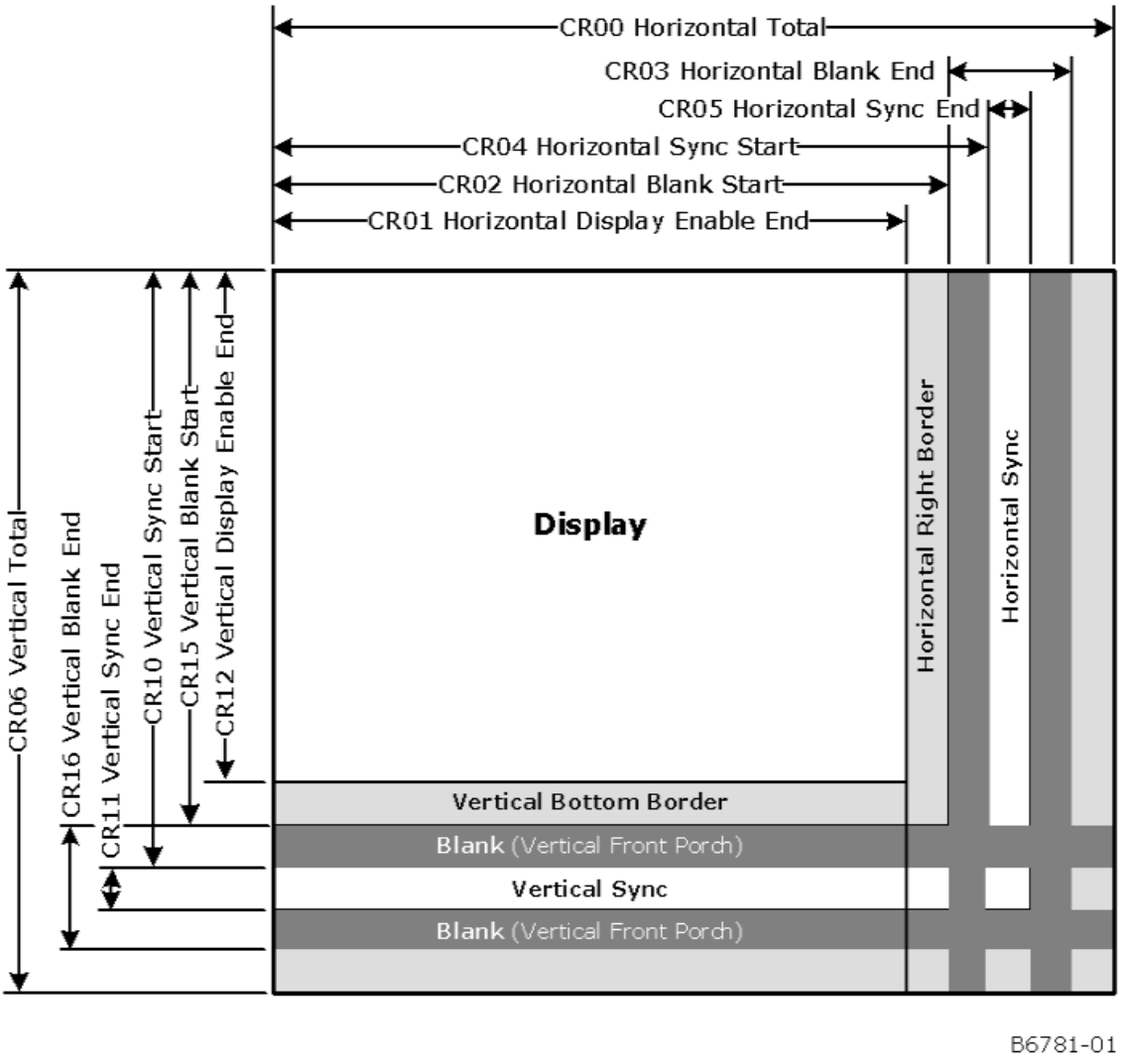
Bit	Description
7:0	**Palette Data.** This byte-wide data port provides read or write access to the three bytes of data of each color data position selected using the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX). The three bytes in each color data position are read or written in three successive read or write operations. The first byte read or written specifies the intensity of the red component of the color specified in the selected color data position. The second byte is for the green component, and the third byte is for the blue component. When writing data to a color data position, all three bytes must be written before the hardware will actually update the three bytes of the selected color data position. When reading or writing to a color data position, ensure that neither the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX) are written to before all three bytes are read or written. A write to either of these two registers causes the circuitry that automatically cycles through providing access to the bytes for red, green and blue components to be reset such that the byte for the red component is the one that will be accessed by the next read or write operation via this register. This register allows access to the palette even when running non-VGA display modes. Writes to the palette can cause sparkle if not done during inactive video periods. This sparkle is caused by an attempt to write and read the same address on the same cycle. Anti-sparkle circuits will substitute the previous pixel value for the read output.

CRT Controller Register

For native VGA modes, the CRTC registers determine the display timing that is to be used. In centered VGA modes, these registers determine the size of the VGA image that is to be centered in the larger timing generator defined rectangle.

The CRT controller registers are accessed by writing the index of the desired register into the CRT Controller Index Register at I/O address 3B4h or 3D4h, depending on whether the graphics system is configured for MDA or CGA emulation. The desired register is then accessed through the data port for the CRT controller registers located at I/O address 3B5h or 3D5h, again depending upon the choice of MDA or CGA emulation as per MSR[0]. For memory mapped accesses, the Index register is at 3B4h (MDA mode) or 3D3h (CGA mode) and the data port is accessed at 3B5h (MDA mode) or 3D5h (CGA mode).

The following figure shows display fields and dimensions and the particular CRxx register that provides the control.



Group 0 Protection: In the original VGA, CR[0:7] could be made write-protected by CR11[7]. In BIOS code, this write protection is set following each mode change. Other protection groups have no current use, and would not be used going forward by the BIOS or by drivers. They are the result of an industry fad some years ago to attempt to write protect other groups of registers; however, all such schemes were chip specific. Only the write protection (Group 0 Protection) is supported.

CRX - CRT Controller Index Register

I/O (and Memory Offset) Address: 3B4h/3D4h

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:0	**CRT Controller Index.** These 7 bits are used to select any one of the CRT controller registers to be accessed via the data port at I/O location 3B5h or 3D5h, depending upon whether the graphics system is configured for MDA or CGA emulation. The data port memory address offsets are 3B5h/3D5h.

CR00 - Horizontal Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=00h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Total.** This register is used to specify the total length of each scan line. This encompasses both the part of the scan line that is within the active display area and the part that is outside of it. Programming this register to a zero has the effect of stopping the fetching of display data. This field should be programmed with a value equal to the total number of character clocks within the entire length of a scan line, minus 5.

CR01 - Horizontal Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=01h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Display Enable End.** This register is used to specify the end of the part of the scan line that is within the active display area relative to its beginning. In other words, this is the horizontal width of the active display area. This field should be programmed with a value equal to the number of character clocks that occur within the horizontal active display area, minus 1. Horizontal display enable will go active at the beginning of each line during vertical active area, it will go inactive based on the programming of this register or the programming of the horizontal total (CR00) register. When this register value is programmed to a number that is larger than the total number of characters on a line, display enable will be active for all but the last character of the horizontal display line.

CR02 - Horizontal Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=02h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Blanking Start.** This register is used to specify the beginning of the horizontal blanking period relative to the beginning of the active display area of a scan line. Horizontal blanking should always be set to start no sooner than after the end of horizontal active. This field should be programmed with a value equal to the number of character clocks that occur on a scan line from the beginning of the active display area to the beginning of the horizontal blanking.

CR03 - Horizontal Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=03h)

Default: 1UUU UUUUb (U=Undefined)

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Reserved.** Values written to this bit are ignored, and to maintain consistency with the VGA standard, a value of 1 is returned when this bit is read. At one time, this bit was used to enable access to certain light pen registers. At that time, setting this bit to 0 provided this access, but setting this bit to 1 was necessary for normal operation.
6:5	**Display Enable Skew Control.** Defines the degree to which the start and end of the active display area are delayed along the length of a scan line to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = no delay 01 = delayed by 1 character clock 10 = delayed by 2 character clocks 11 = delayed by 3 character clocks
4:0	**Horizontal Blanking End Bits [4:0].** This field provides the 5 least significant bits of a 6-bit value that specifies the end of the blanking period relative to its beginning on a single scan line. Bit 7 of the Horizontal Sync End Register (CR05) supplies the most significant bit. This 6-bit value should be programmed to be equal to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02). End of blanking should occur before horizontal total.

CR04 - Horizontal Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=04h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Sync Start** This register is used to specify the position of the beginning of the horizontal sync pulse relative to the start of the active display area on a scan line. This field should be set equal to the number of character clocks that occur from beginning of the active display area to the beginning of the horizontal sync pulse on a single scan line. Horizontal sync should always occur at least 2 clocks after the start of horizontal blank and 2 clocks before the end of horizontal blank. The actual start of sync will also be affected by both the horizontal sync skew register field and whether it is a text or graphics mode.

CR05 - Horizontal Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=05h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Horizontal Blanking End Bit 5.** This bit provides the most significant bit of a 6-bit value that specifies the end of the horizontal blanking period relative to its beginning. Bits [4:0] of Horizontal Blanking End Register (CR03) supplies the 5 least significant bits. See CR03[4:0] for further details. This 6-bit value should be set to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02).
6:5	**Horizontal Sync Delay.** This field defines the degree to which the start and end of the horizontal sync pulse are delayed to compensate for internal pipeline delays. This capability is supplied to implement VGA compatibility. These field describes the delay in terms of a number character clocks. **Bit [6:5] — Amount of Delay** 00 — no delay 01 — delayed by 1 character clock 10 — delayed by 2 character clocks 11 — delayed by 3 character clocks
4:0	**Horizontal Sync End.** This field provides the 5 least significant bits of a 5-bit value that specifies the end of the horizontal sync pulse relative to its beginning. A value equal to the 5 least significant bits of the horizontal character counter value at which time the horizontal retrace signal becomes inactive (logical 0). Thus, this 5-bit value specifies the width of the horizontal sync pulse. To obtain a retrace signal of W, the following algorithm is used: Value of Horizontal Sync start Register (CR04) + width of horizontal retrace signal in character clock units = 5 bit result to be programmed in this field

CR06 - Vertical Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=06h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Vertical Total Bits [7:0].** This field provides the 8 least significant bits of either a 10-bit or 12-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. In standard VGA modes, the vertical total is specified with a 10-bit value. The 8 least significant bits of this value are supplied by these 8 bits of this register, and the 2 most significant bits are supplied by bits 5 and 0 of the Overflow Register (CR07).

CR07 - Overflow Register (Vertical)

I/O (and Memory Offset) Address: 3B5h/3D5h (index=07h)

Default: UU0U UUU0b (U=Undefined)

Attributes: Read/Write (Group 0 Protection on bits [7:5, 3:0])

Bit	Description
7	**Vertical Sync Start Bit 9.** The vertical sync start is a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by this bit and bit 2, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
6	**Vertical Display Enable End Bit 9.** The vertical display enable end is a 10-bit that specifies the number of the last scan line within the active display area. In standard VGA modes, the vertical display enable end is specified with a 10-bit value. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the most and second-most significant bits are supplied by this bit and bit 1, respectively, of this register. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
5	**Vertical Total Bit 9.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by this bit and bit 0, respectively, of this register. This 10-bit value should be programmed equal to the total number of scan lines, minus 2.

Bit	Description
4	**Line Compare Bit 8.** This bit provides the second most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display what data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display what data exists in the frame buffer starting at the first byte of the frame buffer.
3	**Vertical Blanking Start Bit 8.** The vertical blanking start is a 10-bit that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by bit 5 of the Maximum Scan Line Register (CR09) and this bit of this register, respectively. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
2	**Vertical Sync Start Bit 8.** The vertical sync start is a 10-bit value that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by bit 7 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

Bit	Description
1	**Vertical Display Enable End Bit 8.** The vertical display enable end is a 10-bit value that specifies the number of the last scan line within the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the two most significant bits are supplied by bit 6 and this bit, respectively, of this register. This 10-bit or value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
0	**Vertical Total Bit 8.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by bit 5 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the total number of scan lines, minus 2.

CR08 - Preset Row Scan Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=08h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0s.
6:5	**Byte Panning.** This field holds a 2-bit value that selects number of bytes (up to 3) by which the image is shifted horizontally to the left on the screen. This function is available in both text and graphics modes. In text modes with a 9-pixel wide character box, the image can be shifted up to 27 pixels to the left, in increments of 9 pixels. In text modes with an 8-pixel wide character box, and in all standard VGA graphics modes, the image can be shifted up to 24 pixels to the left, in increments of 8 pixels. When the Nine dot disable bit of the VGA control register is set, the pixel shift will be equivalent to the 8-dot mode. The image can be shifted still further, in increments of individual pixels, through the use of bits [3:0] of the Horizontal Pixel Panning Register (AR13).
4:0	**Starting Row Scan Count.** This field specifies which horizontal line of pixels within the character boxes of the characters used on the top-most row of text on the display will be used as the top-most scan line. The horizontal lines of pixels of a character box are numbered from top to bottom, with the top-most line of pixels being number 0. If a horizontal line of the these character boxes other than the top-most line is specified, then the horizontal lines of the character box above the specified line of the character box will not be displayed as part of the top-most row of text characters on the display. Normally, the value specified by these 5 bits should be 0, so that all of the horizontal lines of pixels within these character boxes will be displayed in the top-most row of text, ensuring that the characters in the top-most row of text do not look as though they have been cut off at the top.

Number of Pixels Shifted

Bit [6:5]	9-Pixel Text	8-Pixel Text & Graphics
00	0	0
01	9	8
10	18	16
11	27	24

CR09 - Maximum Scan Line Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=09h)

Default: 00h

Attributes: Read/Write

Bit	Description
7	**Double Scanning Enable.** 0 = Disable. When disabled, the clock to the row scan counter is equal to the horizontal scan rate. This is the normal setting for many of the standard VGA modes. 1 = Enable. When enabled, the clock to the row scan counter is divided by 2. This is normally used to allow CGA-compatible modes that have only 200 scan lines of active video data to be displayed as 400 scan lines (each scan line is displayed twice).
6	**Line Compare Bit 9.** This bit provides the most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 4 of the Overflow Register (CR07) supplies the second most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.
5	**Vertical Blanking Start Bit 9.** The vertical blanking start is a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by this bit and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal to the number of scan line from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.

Bit	Description
4:0	**Starting Row Scan Count.** This field provides all 5 bits of a 5-bit value that specifies the number of scan lines in a horizontal row of text. This value should be programmed to be equal to the number of scan lines in a horizontal row of text, minus 1.

CR0A - Text Cursor Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ah)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5	**Text Cursor Off.** This text cursor exists only in text modes, so this register is entirely ignored in graphics modes. 0 = Enables the text cursor. 1 = Disables the text cursor.
4:0	**Text Cursor Start.** This field specifies which horizontal line of pixels in a character box is to be used to display the first horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the first horizontal line of pixels on which the cursor is to be shown.

CR0B - Text Cursor End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Bh)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Text Cursor Skew.** This field specifies the degree to which the start and end of each horizontal line of pixels making up the cursor is delayed to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = No delay 01 = Delayed by 1 character clock 10 = Delayed by 2 character clocks 11 = Delayed by 3 character clocks
4:0	**Text Cursor End.** This field specifies which horizontal line of pixels in a character box is to be used to display the last horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the last horizontal line of pixels on which the cursor is to be shown.

CR0C - Start Address High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ch)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [15:8].**This register provides either bits 15 through 8 of a 16-bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes, the start address is specified with a 16-bit value. The eight bits of this register provide the eight most significant bits of this value, while the eight bits of the Start Address Low Register (CR0D) provide the eight least significant bits.

CR0D - Start Address Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Dh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [7:0]** This register provides either bits 7 through 0 of a 16 bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes the start address is specified with a 16-bit value. The eight bits of the Start Address High Register (CR0C) provide the eight most significant bits of this value, while the eight bits of this register provide the eight least significant bits.

CR0E - Text Cursor Location High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Eh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [15:8].** This field provides the 8 most significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bit 7:0 of the Text Cursor Location Low Register (CR0F) provide the 8 least significant bits.

CR0F - Text Cursor Location Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Fh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [7:0].** This field provides the 8 least significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bits 7:0 of the Text Cursor Location High Register (CR0E) provide the 8 most significant bits.

CR10 - Vertical Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=10h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Sync Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area of a screen. In standard VGA modes, this value is described in 10 bits with bits [7,2] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should equal the vertical sync start in terms of the number of scan lines from the beginning of the active display area to the beginning of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

CR11 - Vertical Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=11h)

Default: 0U00 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Protect Registers [0:7].** The ability to write to Bit 4 of the Overflow Register (CR07) is not affected by this bit (i.e., bit 4 of the Overflow Register is always writeable). 0 = Enable writes to registers CR[00:07]. (default) 1 = Disable writes to registers CR[00:07].
6	**Reserved.** In the VGA standard, this bit was used to switch between 3 and 5 frame buffer refresh cycles during the time required to draw each horizontal line.
5	**Vertical Interrupt Enable.** This bit is reserved for compatibility only. While this bit may be written or read, it's value will have no effect. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. Bit 7 of Input Status Register 0 (ST00) originally indicated the status of the vertical retrace interrupt. 0 = Enable the generation of an interrupt at the beginning of each vertical retrace period. 1 = Disable the generation of an interrupt at the beginning of each vertical retrace period.
4	**Vertical Interrupt Clear.** This is reserved for compatibility only. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. 0 = Setting this bit to 0 clears a pending vertical retrace interrupt. This bit must be set back to 1 to enable the generation of another vertical retrace interrupt.
3:0	**Vertical Sync End.** This 4-bit field provides a 4-bit value that specifies the end of the vertical sync pulse relative to its beginning. This 4-bit value should be set to the least significant 4 bits of the result of adding the length of the vertical sync pulse in terms of the number of scan lines that occur within the length of the vertical sync pulse to the value that specifies the beginning of the vertical sync pulse (see the description of the Vertical Sync Start Register for more details).

CR12 - Vertical Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=12h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Display Enable End Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the number of the last scan line within the active display area. In standard VGA modes, this value is described in 10 bits with bits [6,1] of the Overflow Register (CR07) supplying the two most significant bits. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area,minus 1.

CR13 - Offset Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=13h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Offset Bits [7:0].** This register provides either all 8 bits of an 8-bit value that specifies the number of words or DWords of frame buffer memory occupied by each horizontal row of characters. Whether this value is interpreted as the number of words or DWords is determined by the settings of the bits in the Clocking Mode Register (SR01). In standard VGA modes, the offset is described with an 8-bit value, all the bits of which are provided by this register. This 8-bit value should be programmed to be equal to either the number of words or DWords (depending on the setting of the bits in the Clocking Mode Register, SR01) of frame buffer memory that is occupied by each horizontal row of characters.

CR14 - Underline Location Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=14h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6	**DWord Mode.** 0 = Frame buffer addresses are interpreted by the frame buffer address decoder as being either byte addresses or word addresses, depending on the setting of bit 6 of the CRT Mode Control Register (CR17). 1 = Frame buffer addresses are interpreted by the frame buffer address decoder as being DWord addresses, regardless of the setting of bit 6 of the CRT Mode Control Register (CR17). This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to select how frame buffer addresses from the CPU are interpreted by the frame buffer address decoder as shown below: **CR14[6] / CR17[6] / Addressing Mode** 0 / 0 / Word Mode 0 / 1 / Byte Mode 1 / 0 / DWord Mode 1 / 1 / DWord Mode
5	**Count By 4.** 0 = The memory address counter is incremented either every character clock or every other character clock, depending upon the setting of bit 3 of the CRT Mode Control Register. 1 = The memory address counter is incremented either every 4 character clocks or every 2 character clocks, depending upon the setting of bit 3 of the CRT Mode Control Register. . This is used in mode x13 to allow for using all four planes. This bit is used in conjunction with bit 3 of the CRT Mode Control Register (CR17) to select the number of character clocks are required to cause the memory address counter to be incremented as shown, below: **CR14[5] / CR17[3] / Addressing Incrementing Interval** 0 / 0 / every character clock 0 / 1 / every 2 character clocks 1 / 0 / every 4 character clocks 1 / 1 / every 2 character clocks

Bit	Description
4:0	**Underline Location.** This field specifies which horizontal line of pixels in a character box is to be used to display a character underline in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the horizontal line on which the character underline mark is to be shown.

CR15 - Vertical Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=15h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Blanking Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area of the screen. In standard VGA modes, the vertical blanking start is specified with a 10-bit value. The most and second-most significant bits of this value are supplied by bit 5 of the Maximum Scan Line Register (CR09) and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal the number of scan lines from the beginning of the active display area to the beginning of the vertical blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which vertical blanking begins.

CR16 - Vertical Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=16h)

Default: Undefined

Attributes: Read/Write

This register provides a 8-bit value that specifies the end of the vertical blanking period relative to its beginning.

Bit	Description
7:0	**Vertical Blanking End Bits [7:0].** This 8-bit value should be set equal to the least significant 8 bits of the result of adding the length of the vertical blanking period in terms of the number of scan lines that occur within the length of the vertical blanking period to the value that specifies the beginning of the vertical blanking period (see the description of the Vertical Blanking Start Register for details).

CR17 - CRT Mode Control

I/O (and Memory Offset) Address: 3B5h/3D5h (index=17h)

Default: 0UU0 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**CRT Controller Reset.** This bit has no effect except in native VGA modes (non-centered). 0 = Forces horizontal and vertical sync signals to be inactive. No other registers or outputs are affected. 1 = Permits normal operation.
6	**Word Mode or Byte Mode.** 0 = The memory address counter's output bits are shifted by 1 bit position before being passed on to the frame buffer address decoder such that they are made into word-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. 1 = The memory address counter's output bits remain unshifted before being passed on to the frame buffer address decoder such that they remain byte-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to control how frame buffer addresses from the memory address counter are interpreted by the frame buffer address decoder as shown below: CR14[6] CR17[6] **Address Mode** 0 0 Word Mode - Addresses from the memory address counter are shifted once to become word-aligned 0 1 Byte Mode - Addresses from the memory address counter are not shifted 1 0 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned 1 1 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned
5	**Address Wrap.** This bit is only effective when word mode is made active by setting bit 6 in both the Underline Location Register and this register to 0. 0 = Wrap frame buffer address at 16 KB. This is used in CGA-compatible modes. 1 = No wrapping of frame buffer addresses.

Bit	Description
4	**Reserved.** Read as 0.
3	**Count By 2.** This bit is used in conjunction with bit 5 of the Underline Location Register (CR14) to select the number of character clocks are required to cause the memory address counter to be incremented. 0 = The memory address counter is incremented either every character clock or every 4 character clocks, depending upon the setting of bit 5 of the Underline Location Register. 1 = The memory address counter is incremented either every other clock. CR14[5] CR17[3] Address Incrementing interval 0 0 every character clock 0 1 every 2 character clocks 1 0 every 4 character clocks 1 1 every 2 character clocks
2	**Horizontal Retrace Select.** This bit provides a way of effectively doubling the vertical resolution by allowing the vertical timing counter to be clocked by the horizontal retrace clock divided by 2 (usually, it would be undivided). 0 = The vertical timing counter is clocked by the horizontal retrace clock. 1 = The vertical timing counter is clocked by the horizontal retrace clock divided by 2.
1	**Select Row Scan Counter.** 0 = A substitution takes place, where bit 14 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 1 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.
0	**Compatibility Mode Support.** 0 = A substitution takes place, where bit 13 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 0 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.

The following tables show the possible ways in which the address bits from the memory address counter can be shifted and/or reorganized before being presented to the frame buffer address decoder. First, the

address bits generated by the memory address counter are reorganized, if need be, to accommodate byte, word or DWord modes. The resulting reorganized outputs (MAOut15-MAOut0) from the memory address counter may also be further manipulated with the substitution of bits from the row scan counter (RSOut1 and RSOut0) before finally being presented to the input bits of the frame buffer address decoder (FBIn15-FBIn0).

Memory Address Counter Address Bits [15:0]

	Byte Mode CR14 bit 6=0 CR17 bit 6=1 CR17 bit 5=X	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=1	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=0	DWord Mode CR14 bit 6=1 CR17 bit 6=X CR17 bit 5=X
MAOut0	0	15	13	12
MAOut1	1	0	0	13
MAOut2	2	1	1	0
MAOut3	3	2	2	1
MAOut4	4	3	3	2
MAOut5	5	4	4	3
MAOut6	6	5	5	4
MAOut7	7	6	6	5
MAOut8	8	7	7	6
MAOut9	9	8	8	7
MAOut10	10	9	9	8
MAOut11	11	10	10	9
MAOut12	12	11	11	10
MAOut13	13	12	12	11
MAOut14	14	13	13	12
MAOut15	15	14	14	13

X = Don't Care

Frame Buffer Address Decoder

	CR17 bit 1=1	CR17 bit 1=1	CR17 bit 1=0	CR17 bit 1=0
	CR17 bit 0=1	CR17 bit 0=0	CR17 bit 0=1	CR17 bit 0=0
FBIn0	MAOut0	MAOut0	MAOut0	MAOut0
FBIn1	MAOut1	MAOut1	MAOut1	MAOut1
FBIn2	MAOut2	MAOut2	MAOut2	MAOut2
FBIn3	MAOut3	MAOut3	MAOut3	MAOut3
FBIn4	MAOut4	MAOut4	MAOut4	MAOut4
FBIn5	MAOut5	MAOut5	MAOut5	MAOut5
FBIn6	MAOut6	MAOut6	MAOut6	MAOut6
FBIn7	MAOut7	MAOut7	MAOut7	MAOut7
FBIn8	MAOut8	MAOut8	MAOut8	MAOut8
FBIn9	MAOut9	MAOut9	MAOut9	MAOut9
FBIn10	MAOut10	MAOut10	MAOut10	MAOut10
FBIn11	MAOut11	MAOut11	MAOut11	MAOut11
FBIn12	MAOut12	MAOut12	MAOut12	MAOut12
FBIn13	MAOut13	MAOut13	RSOut0	RSOut0
FBIn14	MAOut14	RSOut1	MAOut14	RSOut1
FBIn15	MAOut15	MAOut15	MAOut15	MAOut15

CR18 - Line Compare Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=18h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Line Compare Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bit 4 of the Overflow Register (CR07) supplies the second most significant bit. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. (This register is only used in split screening modes, and this is not a problem because split screening is not actually used for extended modes. As a result, there is no benefit to extending the existing overflow bits for higher resolutions.) When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

CR22 - Memory Read Latch Data Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=22h)

Default: 00h

Attributes: Read Only

Bit	Description
7:0	**Memory Read Latch Data.** This field provides the value currently stored in 1 of the four memory read latches. Bits 1 and 0 of the Read Map Select Register (GR04) select which of the four memory read latches may be read via this register.

CR24 - Toggle State of Attribute Controller Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=24h)

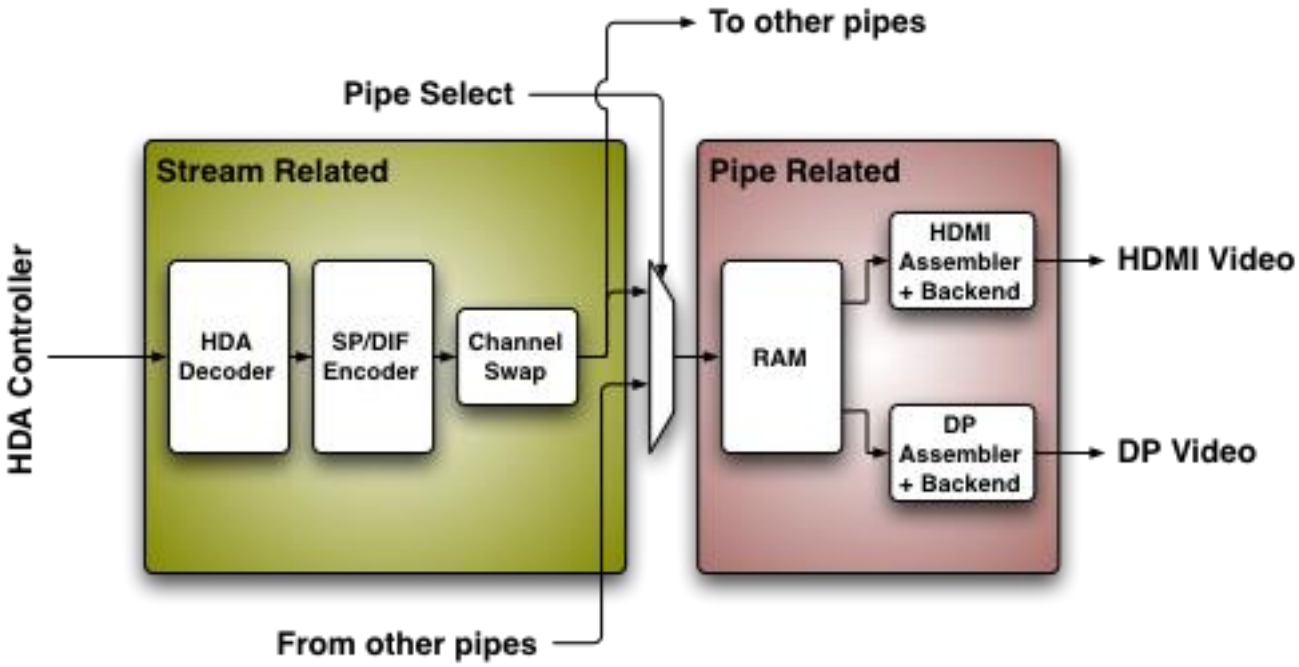
Default: 00h

Attributes: Read Only

Bit	Description
7	**Toggle Status.** Indicates where the last write to attribute register was to: 0 = index port 1 = data port
6:0	**Reserved.** Read as 0.

Display Audio Codec Verbs

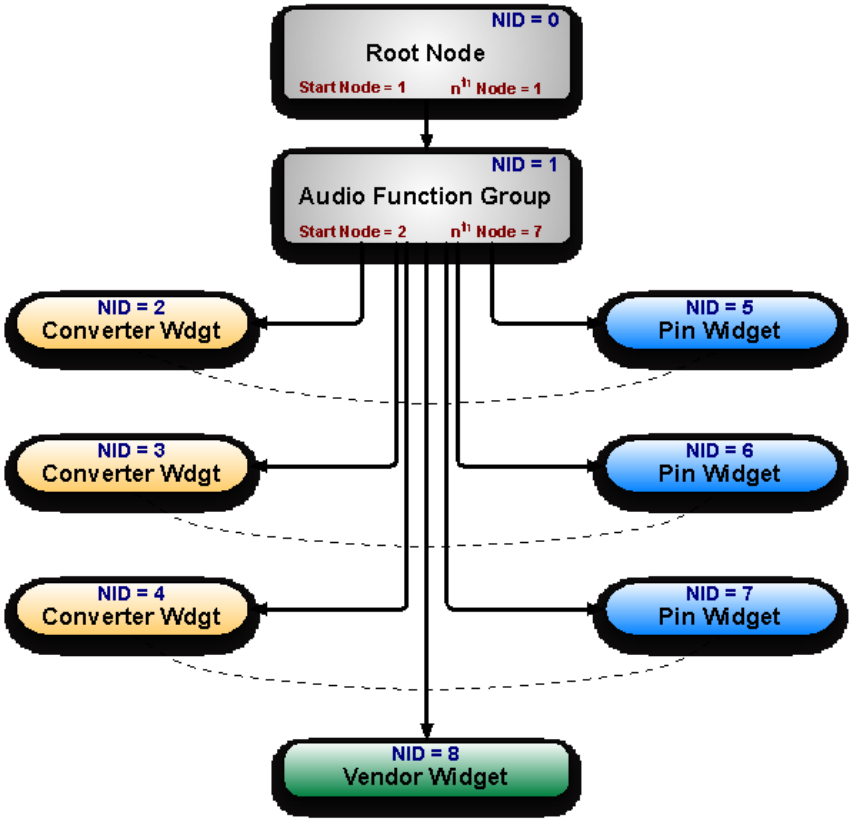
Block Diagram



Codec Node Hierarchy

The diagram below shows the hierarchy of the internal codec. The codec is presented as a single codec with multiple endpoints. By operating as a single codec, only one driver needs to be loaded on the system.

Inside the codec are three "converter widgets" and three "pin widgets", responsible for taking data from HD Audio DMA engines and placing into an HDMI/DP stream. Each pin widget has a 1-1 connection to a converter widget (as indicated by the dotted lines in the diagram).



Programming

Programming of the codec is performed by "verbs" as described in the HD Audio specification. These verbs travel over the internal HD Audio link at a rate of 1 verb per frame. A verb can either come from the CORB, with responses using the RIRB, or using an immediate command and response mechanism (ICR). Device 2 contains its own copy of an ICR mechanism as a back-door into the audio codec.

Verb Support

Verb ID		Verb Name/Description	Node ID							
Set	Get		01h	02h	03h	04h	05h	06h	07h	08h
2h	Ah	Stream Descriptor Format		Y	Y	Y				
3h	Bh	Set Amplifier Mute					Y	Y	Y	
-	F00h	Get Parameters	Y	Y	Y	Y	Y	Y	Y	Y
701h	F01h	Connection Select Control					Y	Y	Y	
-	F02h	Connection List Entry					Y	Y	Y	
705h	F05h	Power State		Y	Y	Y	Y	Y	Y	
706h	F06h	Channel and Stream ID		Y	Y	Y				
707h	F07h	Pin Widget Control					Y	Y	Y	
708h	F08h	Unsolicited Response Enable					Y	Y	Y	Y
-	F09h	Pin Sense					Y	Y	Y	
-	F0Dh	Digital Converter		Y	Y	Y				
70Dh	-	Digital Converter 1		Y	Y	Y				
70Eh	-	Digital Converter 2		Y	Y	Y				
-	F1Ch	Configuration Default					Y	Y	Y	
71Ch	-	Configuration Default Byte 0					Y	Y	Y	
71Dh	-	Configuration Default Byte 1					Y	Y	Y	
71Eh	-	Configuration Default Byte 2					Y	Y	Y	
71Fh	-	Configuration Default Byte 3					Y	Y	Y	
-	F20h	Subsystem ID	Y							
-	F21h	Subsystem ID	Y							
-	F22h	Subsystem ID	Y							
-	F23h	Subsystem ID	Y							
720h	-	Subsystem ID[7: 0]	Y							
721h	-	Subsystem ID[15: 8]	Y							
722h	-	Subsystem ID[23:16]	Y							
723h	-	Subsystem ID[31:24]	Y							
72Dh	F2Dh	Converter Channel Count		Y	Y	Y				
-	F2Eh	HDMI/DP Info Size					Y	Y	Y	

Verb ID		Verb Name/Description	Node ID							
Set	Get		01h	02h	03h	04h	05h	06h	07h	08h
730h	F30h	HDMI Info Index					Y	Y	Y	
731h	F31h	HDMI Info Data					Y	Y	Y	
732h	F32h	HDMI Info Transmit Control					Y	Y	Y	
734h	F34h	Converter Channel Map					Y	Y	Y	
735h	F35h	Device Select					Y	Y	Y	
-	F36h	Display Device List Entry					Y	Y	Y	
73Ch	73Ch	DisplayPort Stream ID					Y	Y	Y	
73Eh	-	Digital Converter 3		Y	Y	Y				
73Fh	-	Digital Converter 4		Y	Y	Y				
-	F80h	HDMI / DP Status								Y
781h	F81h	HDMI Vendor Verb								Y
782h	-	GTC Capture Trigger								Y
-	F83h	Captured Wall Clock Value								Y
-	F84h	Captured GTC Value								Y
-	F85h	Get GTC Offset Value								Y
785h	-	Set GTC Offset Value[7: 0]								Y
786h	-	Set GTC Offset Value[15: 8]								Y
787h	-	Set GTC Offset Value[23:16]								Y
788h	-	Set GTC Offset Value[31:24]								Y
789h	F89h	Converter Channel Count								Y

Parameter Support

Param ID	Parameter Name	Node ID								
		00h	01h	02h	03h	04h	05h	06h	07h	08h
00h	Vendor ID	Y								
02h	Revision ID	Y								
04h	Subordinate Node Count	Y	Y							
05h	Function Group Type		Y							
08h	Audio Function Group Capabilities									
09h	Audio Widget Capabilities			Y	Y	Y	Y	Y	Y	Y
0Ah	Sample Size, Rate CAPs			Y	Y	Y				
0Bh	Stream Formats			Y	Y	Y				
0Ch	Pin Capabilities						Y	Y	Y	
0Dh	Input Amp Capabilities									
0Eh	Connection List Length						Y	Y	Y	
0Fh	Supported Power States		Y							
10h	Processing Capabilities									
11h	GPIO Count									
12h	Output Amp Capabilities						Y	Y	Y	
13h	Volume Knob Capabilities									
15h	Device List Length						Y	Y	Y	

Node ID 00h Root Node Verbs

The root node only contains a single verb - the "Get Parameters" verb at F00h.

F00h - Get Parameters

Parameter	Symbol	Register Name
00h	PARAM_VID	Vendor ID
02h	PARAM_RID	Revision ID
04h	PARAM_SNC	Subordinate Node Count

Parameter 00h: VID - Vendor ID

Bit	Reset	Description
31:16	8086h	**Vendor ID (VID):** Indicates the 16-bit Vendor ID values used to identify the codec to the PnP subsystem.
15:00	2809h	**Device ID (DID):** Indicates the 16-bit Device ID values used to identify the codec to the PnP subsystem. It is 280Bh for KBL

Parameter 02h: RID - Revision ID

Bit	Reset	Description
31:24	*0*	*Reserved*
23:20	1h	**Major Revision (MJR):** Indicates the major revision number (left of the decimal) of the High Definition Audio Specification to which the codec is fully compliant.
19:16	0h	**Minor Revision (MNR):** Indicates the minor revision number (right of the decimal) or "dot number" of the High Definition Audio Specification to which the codec is fully compliant.
15:08	00h	**Revision ID (RID):** Indicates the vendor's revision number for this given Device ID.
07:00	00h	**Stepping ID (SID):** Indicates optional vendor stepping number within the revision.

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	0h	**Starting Node Number (SNN):** Indicates the first sub-node's ID is 01h.
15:08	00h	Reserved
07:00	01h	**Total Number of Nodes (TNN):** Indicates one sub-node

F37h GET CCF - Get Current Clock Frequency

Bits	Default	Description
31:6	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz. Reserved for Display Codec
04	1	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz. Reserved for Display Codec
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz. Reserved for Display Codec
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz. Reserved for Display Codec

Node ID 01h Audio Function Group Verbs

Set Verb	Get Verb	Symbol	Name
-	F00h	GET PARAM	Get Parameters
705h	F05h	SET_PS / GET_PS	Set Power State
-	F20h	GET_SSID	Get Subsystem ID
720h	F20h	SET_SSID0	Set/Get Subsystem ID
721h	F21h	SET_SSID1	Set/Get Subsystem ID
722h	F22h	SET_SSID2	Set//Get Subsystem ID
723h	F23h	SET_SSID3	Set/Get Subsystem ID
724h	F24h	SET CCF	Set/Get Current Clock Frequency

F00h Get Parameters

Parameter	Symbol	Register Name
04h	PARAM_SNC	Subordinate Node Count
05h	PARAM_FGT	Function Group Type
08h	PARAM_FGC	Function Group Capabilities
0Fh	PARAM SPS	Supported Power States

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	02h	**Start Node Number (SNN):** Indicates the start node number of widget or functional nodes in the Functional Group.
15:08	*0*	*Reserved*
07:00	07h	**Total Number of Nodes (TNN):** Indicates 7 widgets in the Functional Group. (HDMI/DP converters (3) + HDMI/DP pins (3) + Vendor Defined Widget (1)).

Parameter 05h: PARAM_FGT - Function Group Type

Bit	Reset	Description
31:09	*0*	*Reserved*
08	0	**Unsolicited Capable (UC):** Not capable of generating an unsolicited response.
07:00	01h	**Node Type (NT):** Indicates Audio Function Group.

Parameter 08h: PARAM_FGC - Function Group Capability

Bit	Reset	Description
31:04	*0*	*Reserved*
03:00	00h	**Output Delay (OD)**Output Delay.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30	1	**Clock Stop (CS):** Indicates support for D3 when clock is stopped.
29:04	*0*	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

Parameter 16h: PARAM_A2CAP - Azalia 2 Capabilities

Bits	Reset	Description
31:17	*0*	*Reserved*
16	0	**Independent Codec Clock (ICC):** When set, this indicates that the codec generates its own clock, which may drift from the link clock. When cleared, the codec's clock is locked to the link clock.
15:06	*0*	*Reserved*
05	*0*	*Reserved for 192 MHz support.*
04	1	**96MHz Supported (S96):** Indicates 96 MHz clock is supported.
03	1	**48MHz Supported (S48):** Indicates 48 MHz clock is supported.This bit must always be set.
02	0	**24MHz Supported (S24):** Indicates 24 MHz clock is supported. Reserved for Display Codec
01	0	**12MHz Supported (S12):** Indicates 12 MHz clock is supported. Reserved for Display Codec
00	0	**6MHz Supported (S6)**Indicates 6 MHz clock is supported. Reserved for Display Codec

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	1	**Clock Stop OK (CSOK):** Clock stopping in D3 is OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

F20h GET SSID - Get Subsystem ID0

Bits	Reset	Description
31:00	80860101h	**Subsystem ID (SSID):** Reports the sub-system ID set via SET_SSIDx verbs.

720h SET SSID0 - Set Subsystem ID0

Bits	Description
07:00	Subsystem ID Bits [7:0]

721h SET SSID1 - Set Subsystem ID1

Bits	Description
07:00	Subsystem ID Bits [15:8]

722h SET SSID2 - Set Subsystem ID2

Bits	Description
07:00	Subsystem ID Bits [23:16]

723h SET SSID3 - Set Subsystem ID3

Bits	Description
07:00	Subsystem ID Bits [31:24]

724h SET CCF - Set Current Clock Frequency

Bits	Description
07:06	Reserved
05	**Set Clock 192 MHz (S192)**: Set clock to 192 MHz.
04	**Set Clock 96 MHz (S96)**: Set clock to 96 MHz
03	**Set Clock 48 MHz (S48)**: Set clock to 48 MHz
02	**Set Clock 24 MHz (S24)**: Set clock to 24 MHz
01	**Set Clock 12 MHz (S12)**: Set clock to 12 MHz
00	**Set Clock 6 MHz (S6)**: Set clock to 6 MHz

F24h GET CCF - Get Current Clock Frequency

Bits	Bits	Description
31:06	0	*Reserved*
05	0	**Current Clock 192 MHz (C192)**: Indicates the current clock is 192 MHz.
04	0	**Current Clock 96 MHz (C96)**: Indicates the current clock is 96 MHz.
03	0	**Current Clock 48 MHz (C48)**: Indicates the current clock is 48 MHz.
02	0	**Current Clock 24 MHz (C24)**: Indicates the current clock is 24 MHz.
01	0	**Current Clock 12 MHz (C12)**: Indicates the current clock is 12 MHz.
00	0	**Current Clock 6 MHz (C6)**: Indicates the current clock is 6 MHz.

7FFh SET Function Group Reset

Bits	Reset	Description
07:00	00h	The Function Reset command causes the functional unit, and all widgets associated with the functional unit, to return to their power-on reset values. Note that some controls such as the Configuration Default controls should not be reset with this command. It is also possible that certain other controls, such as Caller-ID, should not be reset. This command does not affect the Link interface logic, which must be reset with the link RST# signal. Therefore, a codec must not initiate a Status Change request on the link.

Node ID 02h 03h 04h Audio Output Convertor Widget Verbs

Verb	Symbol	Verb Name
2h	SET_SDF	Set Stream Descriptor Format
Ah	GET_SDF	Get Stream Descriptor Format
F00h	GET_PARAM	Get Parameters
705h	SET_PS	Set Power State
F05h	GET_PS	Get Power State
706h	SET_CSID	Set Channel and Stream ID
F06h	GET_CSID	Get Channel and Stream ID
F0Dh	SET_DC1	Get Digital Converter
70Dh	SET_DC1	Set Digital Converter 1

Verb	Symbol	Verb Name
70Eh	SET_DC2	Set Digital Converter 2
73Eh	SET_DC3	Set Digital Converter 3
73Fh	SET_DC4	Set Digital Converter 4
72Dh	SET_CCC	Set Converter Channel Count
F2Dh	GET_CCC	Get Converter Channel Count

2hAh SETGET_SDF - SetGET Stream Descriptor Format

Bits	Reset	Description
31:15	0	Reserved
14	0	**Sample Base Rate (SBR):**
13:11	000	**Sample Base Rate Multiplier (SBRM):**
10:08	000	**Sample Base Rate Divisor (SBRD):**
07	0	Reserved
06:04	011	**Bits / Sample (BPS):** • 001b: Data is packed in memory in 16 bit containers on 16 bit boundaries • 010b: Data is packed in memory in 20 bit containers on 32 bit boundaries • 011b: Data is packed in memory in 24 bit containers on 24 bit boundaries • 100b: Data is packed in memory in 32 bit containers on 32 bit boundaries All other bit combinations reserved
03:00	1h	**# Channels in Stream (NCS):** 2 channels in each frame

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ah	PARAM_PSB	Parameter Sizes and Bit Rates
0Bh	PARAM_SF	Stream Formats
0Fh	PARAM_SPS	Power Supported States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	0h	**Widget Type (TYPE):** Indicates this is an audio output widget
19:16		Sample Delay in Widget (DELAY):
15:13	011	**Channel Count Extension (CCE):** These three bits, combined with STRO, indicate that there are 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	0	**Connection List (CL):** Indicates no connection list
07	0	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	1	**Format Override (FO):** Indicates support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier support.
02	0	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ah: PSB - PCM Sizes and Bit Rates

Bits	Reset	Description
31:21	0	*Reserved*
20	1	**32-bit Support (B32):** Indicates 32-bit samples supported
19	1	**24-bit Support (B24):** Indicates 24-bit samples supported
18	0	**20-bit Support (B20):** Indicates 20-bit samples supported
17	1	**16-bit Support (B16):** Indicates 16-bit samples supported
16	0	**8-bit Support (B8):** Indicates 8-bit samples not supported
15:12	0	*Reserved*
11	0	**384 kHz Support (R12):** Indicates 384 kHz not supported
10	1	**192 kHz Support (R11):** Indicates 192 kHz supported
09	1	**176.4 kHz Support (R10):** Indicates 176.4 kHz supported
08	1	**96 kHz Support (R9):** Indicates 96 kHz supported
07	1	**88.2 kHz Support (R8):** Indicates 88.2 kHz supported
06	1	**48 kHz Support (R7):** Indicates 48 kHz supported
05	1	**44.1 kHz Support (R6):** Indicates 44.1 kHz supported
04	1	**32 kHz Support (R5):** Indicates 32 kHz supported
03	0	**22.05 kHz Support (R4):** Indicates 22.05 kHz not supported
02	0	**16 kHz Support (R3):** Indicates 16 kHz not supported
01	0	**11.025 kHz Support (R2):** Indicates 11.025 kHz not supported
00	0	**8 kHz Support (R1):** Indicates 8 kHz not supported

Parameter 0Bh: SF - Stream Formats

Bits	Reset	Description
31:03	0	*Reserved*
02	1	**AC3 Support (AC3):** Indicates AC3 stream format is supported
01	0	**Float32 Support (F32):** Indicates float32 stream format not supported
00	1	**PCM Support (PCM):** Indicates PCM format is supported.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** ???
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK

Bits	Reset	Description
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

706hF06h GETSET_CSID - GetSet Channel and Stream ID

Bits	Reset	Description
07:04	0h	**Stream ID (SID):** Link stream used by the converter for data output.
03:00	0h	**Lowest Channel Number (LCN):** Lowest channel used by the converter.

Digital Converter Verbs

F0Dh: GET_DC - Get Digital Converter

Bits	Reset	Description
31:24	0	Reserved
23	1	**Keep Alive (KA):** See SET_DC3.KA
22:20	0	Reserved
19:16	0h	**IEC Coding Type (ICT):** See SET_DC3.ICT
15	0	Reserved
14:08	00h	**Category Code (CC):** See SET_DC1.CC
07	0	**Level (LVL):** See SET_DC1.LVL
06	0	**Professional (PRO):** See SET_DC1.PRO
05	0	**Audio is not PCM (AUDIO):** See SET_DC1.AUDIO
04	0	**Copyright (COPY):** See SET_DC1.COPY
03	0	**Pre-emphasis (PRE):** See SET_DC1.PRE

Bits	Reset	Description
02	0	**Validity Configuration (VCFG):** See SET_DC1.VCFG
01	0	**Validity (V):** See SET_DC1.V
00	1	**Digital Enable (DIGEN):** See SET_DC1.DIGEN

70Dh: SET_DC1 - Set Digital Converter 1

Bits	Description
07	**Level (LVL):** S/PDIF IEC Generation Level.
06	**Professional (PRO):** When set, indicates professional use of channel.
05	**Audio is not PCM (AUDIO):** When set, data is non-PCM format.
04	**Copyright (COPY):** When set, copyright asserted.
03	**Pre-emphasis (PRE):** When set, enables filter pre-emphasis.
02	**Validity Configuration (VCFG):** Determines S/PDIF transmitter behavior when data is not being transmitted.
01	**Validity (V):** Affects the validity flag transmitted in each sub-frame, and enables S/PDIF transmitter to maintain connection during error or mute conditions.
00	**Digital Enable (DIGEN):** When set, enables digital content

70Eh: Digital Converter 2

Bits	Description
07	Reserved
06:00	**Category Code (CC):** S/PDIF IEC Category Code.

73Eh: Digital Converter 3

Bits	Description
07	Keep Alive
06:04	Reserved
03:00	IEC Coding Type

73Fh: Digital Converter 4

Bits	Description
07:00	Reserved

72DhF2Dh GETSET_CCC - GetSet Converter Channel Count

Bits	Reset	Description
07:04	0	Reserved
03:00	0000	Converter Channel Count 1 (0th order)

Node ID 05h 06h 07h Pin Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
3h	-	SET_AM	Set Amplifier Mute
-	Bh	GET_AM	Get Amplifier Mute
-	F00h	-	Get Parameters
701h	F01h	SET_CSC / GET_CSC	Set/Get Connection Select Control
-	F02h	-	Get Connection List Entry
705h	F05h	SET_PS / GET_PS	Set/Get Power State
707h	F07h	SET_PWC / GET_PWC	Set/Get Pin Widget Control
708h	F08h	SET_UE / GET_UE	Set/Get Unsolicited Response Enable
-	F09h	-	Get Pin Sense
71Ch	-	SET_CD0	Set Configuration Default Byte 0
71Dh	-	SET_CD1	Set Configuration Default Byte 1
71Eh	-	SET_CD2	Set Configuration Default Byte 2
71Fh	-	SET_CD3	Set Configuration Default Byte 3
-	F1Ch	GET_CD	Get Configuration Default
-	F2Eh	GET_HDIS	Get HDMI/DP Info Size
730h	F30h	SET_HII / GET_HII	Set/Get HDMI Info Index
731h	F31h	SET_HID / GET_HID	Set/Get HDMI Info Data
732h	F32h	SET_HITC / GET_HITC	Set/Get HDMI Info Transmit Control
733h	F33h	SET_PC / GET_PC	Set/Get Protection Control
734h	F34h	SET_CCM / GET_CCM	Set/Get Converter Channel Map
735h	F35h	SET_DS / GET_DS	Set/Get Device Select
-	F36h	GET_DDLE	Get Display Device List Entry
73Ch	F3Ch	SET_DPID / GET_DPID	Set/Get DisplayPort Stream ID

3h SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

B8h GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	0	*Reserved*
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
15h	PARAM_DLL	Device List Length
0Fh	PARAM_SPS	Supported Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	0	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio

Bits	Reset	Description
26	0	*Reserved*
25	1	**Multi Stream Capable (MSC):** Indicates support for DisplayPort Multistream. Will be zero in vanilla mode.
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	0	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI
06:05	0	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	0	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	0	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	0	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 15h: DLL - Device List Length

Bits	Reset	Description
31:06	0	*Reserved*
05:00	00h	**Length (LEN):** Indicates no devices

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

701hF01h SETGET_CSC - SetGet Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC):**

F02h GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:08	0	Reserved
07:00	Varies	**Connection List Entry (CLE):** 02h for NodeID 05h, 03h for NodeID 06h, and 04h for NodeID 07h.

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707hF07h SETGET_PWC - SetGet Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT):**

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG):**

F09h GET_PS - Get Pin Sense

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV):**
29	0	**Inactive (INA):**
28:00	28:00	Reserved

71Ch SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	Default Association (DA):
03:00	Sequence (SEQ):

71Dh SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	Color (COL):
03:00	Miscellaneous (MISC):

71Eh SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	Default Device (DD):
03:00	Connection Type (CT):

71Fh SET_CD3 - Set Configuration Default Byte 3

Bits	Description
07:06	**Port Connectivity (PC):** External connectivity of the pin complex. • 00 = Connected to jack • 01 = No physical connection • 10 = Fixed function device (integrated speaker, mic, etc.) • 11 = Both a jack and internal connection
05:00	**Location (LOC):**

Bits	Description

	Bits 3:0										
Bits 5:4	0h: N/A	1h: Rear	2h: Front	3h: Left	4h: Right	5h: Top	6h: Bottom	7h: Special	8h: Special	9h: Special	Ah-Fh Reserved
00: External	Y	Y	Y	Y	Y	Y	Y	Y	Y		
01: Internal	Y							Y	Y	Y	
10: Separate Chassis	Y	Y	Y	Y	Y	Y	Y				
11: Other	Y						Y	Y	Y		

F1Ch GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Eh HDMIDP Info Size

Bits	Reset	Description
31:08	0	Reset
07:00	Varies	**Size (SZ):** Indexes 0 - 3 return 1Eh, index 1000 returns 53h, others reserved.

F2Fh Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

730hF30h SETGET_HII - SetGet HDMI Info Index

Bits	Reset	Description
07:05	000	**Infoframe Packet Index (IPI):** Value / Name / Value / Name 000 / Audio / 011 / GP3 001 / GP / 100 / GP4 010 / GP2 / Others / Reserved
04:00	00h	**Byte Offset Index Pointer (BOI):**

731hF31h SETGET_HID - SetGet HDMI Info Data

Bits	Reset	Description
07:00	00h	**Data (DATA):** Data at current index pointed to from SET_HII verb.

732hF32h SETGET_HITC - SetGet HDMI Info Transmit Control

Bits	Reset	Description
07:06	00	**InfoFrame Control Current Indexed Frame (IFCCIF):** • 00 = Disable Transmit • 01 = Reserved • 10 = Transmit Once • 11 = Best Effort
05:00	0	*Reserved*

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734hF34h SETGET_CCM - GetSet Converter Channel Map

Bits	Reset	Description
07:04	0h	Converter Channel (CC):
03:00	0h	Slot (SN):

735h SET_DS - Set Device Select

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present)

F35h: GET_DS - Get Device Select

Bits	Description
31:12	**Reserved:** Set to 0
11:06	**SinK Device ID:** Sink Device ID in the multi stream topology of the DP hierarchy. Device attached to Pipe A will have ID of "00000", PipeB will have "00001" and Pipe C will have "00010".
05:00	**Device (D):**Device Entry index currently set

F36h GET_DDLE - Get Display Device List Entry

Bits	Bits	Description
31:12	0	*Reserved*
11	0	**Reserved?**
10	0	**IA of Entry 2**
09	0	**ELDV of Entry 2**

Bits	Bits	Description
08	0	**PD of Entry 2**
07	0	**Reserved?**
06	0	**IA of Entry 1**
05	0	**ELDV of Entry 1**
04	0	**PD of Entry 1**
03	0	**Reserved?**
02	0	**IA of Entry 0**
01	0	**ELDV of Entry 0**
00	0	**PD of Entry 0**

73ChF3Ch SETGET_DPID - SetGet DisplayPort Stream ID

Bits	Reset	Description
07:03	00h	**Tag (TAG):** Represents the SSID that will go in the lower 5 bits of the SSID
02:00	000	**Index (IDX):** Pointer to program multiple SSID

Node ID 08h Intel Vendor Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
-	F00h	GET_PARAM	Get Parameters
-	F80h	GET_HDPS	Get HDMI/DP Status
781h	F81h	SET_HVV / GET_HVV	Set/Get iDisp Codec Vendor Verb
782h	-	SET_GTCT	Set GTC Trigger
-	F83h	GET_CWC	Get Captured Wall Clock
	F84h	GET_CGTC	Get Captured GTC Value
	F85h	GET_GOF	Get GTC Offset Value
785h	-	SET_GOF0	Set GTC Offset Value Byte 0
786h	-	SET_GOF1	Set GTC Offset Value Byte 1
787h	-	SET_GOF2	Set GTC Offset Value Byte 2
788h	-	SET_GOF3	Set GTC Offset Value Byte 3

Set Verb	Get Verb	Symbol	Verb Name
789h	F89h	SET_GDI / GET_GDI	Set/Get GTC Offset Device Index

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	Fh	**Widget Type (TYPE):** Indicates this is a vendor defined widget
19:00	0	Reserved

71Eh SET_GET_GFXMAILBOX - Set Get GFX MAILBOX Byte 2

71Eh: SET GFX MAILBOXM

Bits	Default	Description
07:00	00	Contents to be defined by GFX driver and audio driver

F1Eh: GET GFX MAILBOX

Bits	Default	Description
31:24	00h	Other values of 71F, verbs
23:16	00h	Contents to be defined by GFX driver and audio driver
15:00	0000h	Other values of 71D, 71C verb

728h SET CLOCK OFF - Set Clock Off Command

Bits	Description
07:00	Data is Irrelevant

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses from GFX MAIL BOX register writes are allowed.
06	Reserved
05:00	**Tag (TAG):** Tag for GFX Mail box register unsol responses.

781hF81h GETSET_VV - GetSet iDisp Codec Vendor Verb

Bits	Bits	Description
07:06	0	**Port Select:** This filed is programmed to select the port to be exposed to the inbox driver in the vanilla mode. Will not reset on double function group reset. 00 -> Port B 01 -> Port C 10 -> Port D 11 -> Reserved
05:04	0	**Reserved**
03	0	**Enable WFA:** When Set, WFA mode is enabled**.** Will not reset on double function group reset.
02	0	**Reserved**
01	0	**Enable DP1.2 Features (EDP12):** When set, DP1.2 features are enabled.Will not reset on double function group reset.
00	0	**Enable 3rd Pin and Converter Widget (E3P):** When set, the second and third pin and converter widget is enabled and can respond to HD Audio Verbs. When cleared, the second and third pin and converter widget is disabled and cannot respond to HD Audio verbs.Will not reset on double function group reset.

782h SET_GTCT - Set GTC Trigger

Bits	Bits	Description
07:00	0	**Any data:** The value of this field is irrelevant. The access of the SET causes a capture to occur.

F83h GET_CGTC - Get Captured GTC Value

Bits	Bits	Description
07	0	**GTC Value:** 32-bit GTC value captured on the SET_GTCT verb

F84h GET_CWC - Get Captured Wall Clock Value

Bits	Bits	Description
31:00	0	**Wall Clock Value:** 32-bit wall clock value captured on the SET_GTCT verb

F85h GET GOF - Get GTC Offset Value

Bits	Reset	Description
31:00	0h	**Value (VAL):** Reports the GTC Offset Value.

785h SET GOF0 - Set GTC Offset Value Byte 0

Bits	Description
07:00	GTC Offset Value Bits [7:0]

786h SET GOF1 - Set GTC Offset Value Byte 1

Bits	Description
07:00	GTC Offset Value Bits [15:8]

787h SET GOF2 - Set GTC Offset Value Byte 2

Bits	Description
07:00	GTC Offset Value Bits [23:16]

788h SET GOF3 - GTC Offset Value Byte 3

Bits	Description
07:00	GTC Offset Value Bits [31:24]

789hF89h SETGET_GDI - SetGet GTC Device Index

Bits	Reset	Description
07:06	0	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present) This is the pipe based. 000000 i s pipeA and so on and so forth.

3h SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.

Bits	Bits	Description
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

B8h GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	0	*Reserved*
07	1	**Mute (MUTE):** When set, amp muted.
06:00	0	*Reserved*

F00h Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
0Fh	PARAM_PS	Suppoted Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.

Bits	Reset	Description
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	0	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio
26	0	*Reserved*
25	0	*MST capable (DP1.2)*
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	0	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI

Bits	Reset	Description
06:05	0	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	0	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	0	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	0	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.

Bit	Reset	Description
00	1	**D0 Supported (D0S):** Indicates support for D0.

701hF01h SETGET_CSC - SetGet Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC)**

F02h GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:24	0	Reserved
23:16	100	**Connection List Entry (CLE):** Hardwired to Converter ID 4
15:08	011	**Connection List Entry (CLE)**: Hardwired to Converter ID 3
07:00	010	**Connection List Entry (CLE)**: Hardwired to Converter ID 2

705h SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.

Bits	Reset	Description
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707hF07h SETGET_PWC - SetGet Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT)**

708hF08h SETGET_UE - SetGet Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG)**

F09h GET_PS - Get Pin Sense

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV)**
29	0	**Inactive (INA)**
28:00	28:00	Reserved

71Ch SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	**Default Association (DA)**
03:00	**Sequence (SEQ)**

71Dh SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	**Color (COL)**
03:00	**Miscellaneous (MISC)**

71Eh SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	**Default Device (DD)**
03:00	**Connection Type (CT)**

71Fh SET_CD3 - Set Configuration Default Byte 3

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>07:06</td><td>Port Connectivity (PC): External connectivity of the pin complex.
00 = Connected to jack
01 = No physical connection
10 = Fixed function device (integrated speaker, mic, etc.)
11 = Both a jack and internal connection</td></tr>
<tr><td>05:00</td><td>Location (LOC):
<table>
<tr><td></td><td colspan="10">Bits 3:0</td></tr>
<tr><td>Bits 5:4</td><td>0h: N/A</td><td>1h: Rear</td><td>2h: Front</td><td>3h: Left</td><td>4h: Right</td><td>5h: Top</td><td>6h: Bottom</td><td>7h: Special</td><td>8h: Special</td><td>9h: Special</td><td>Ah-Fh Reserved</td></tr>
<tr><td>00: External</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
<tr><td>01: Internal</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td></tr>
<tr><td>10: Separate Chassis</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11: Other</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
</table></td></tr>
<tr><td></td><td></td></tr>
</table>

F1Ch GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Fh Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734hF34h SETGET_CCM - GetSet Converter Channel Map

Bits	Reset	Description
07:04	0h	**Converter Channel (CC)**
03:00	0h	**Slot (SN)**

740hF40h SETGET PTS Offset Byte0

Bits	Reset	Description
07:00	00h	**PTS B0(PTSB0):** Bits 7:0 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

741hF41h SETGET PTS Offset Byte1

Bits	Reset	Description
07:00	00h	**PTS B1(PTSB1):** Bits 15:8 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

742hF42h SETGET PTS Offset Byte2

Bits	Reset	Description
07:00	00h	**PTS B2(PTSB2):** Bits 23:16 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

743hF43h SETGET PTS Offset Byte3

Bits	Reset	Description
07:00	00h	**PTS B3(PTSB3):** Bits 31:24 of the 32 bit 2's complement offset value. Should be added to the PTS value before sending to the circular buffer in the PES packet. Get verb will give the 32 bits value

North Display Engine Registers

This chapter contains the register descriptions for the display portion of a family of graphics devices.

These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

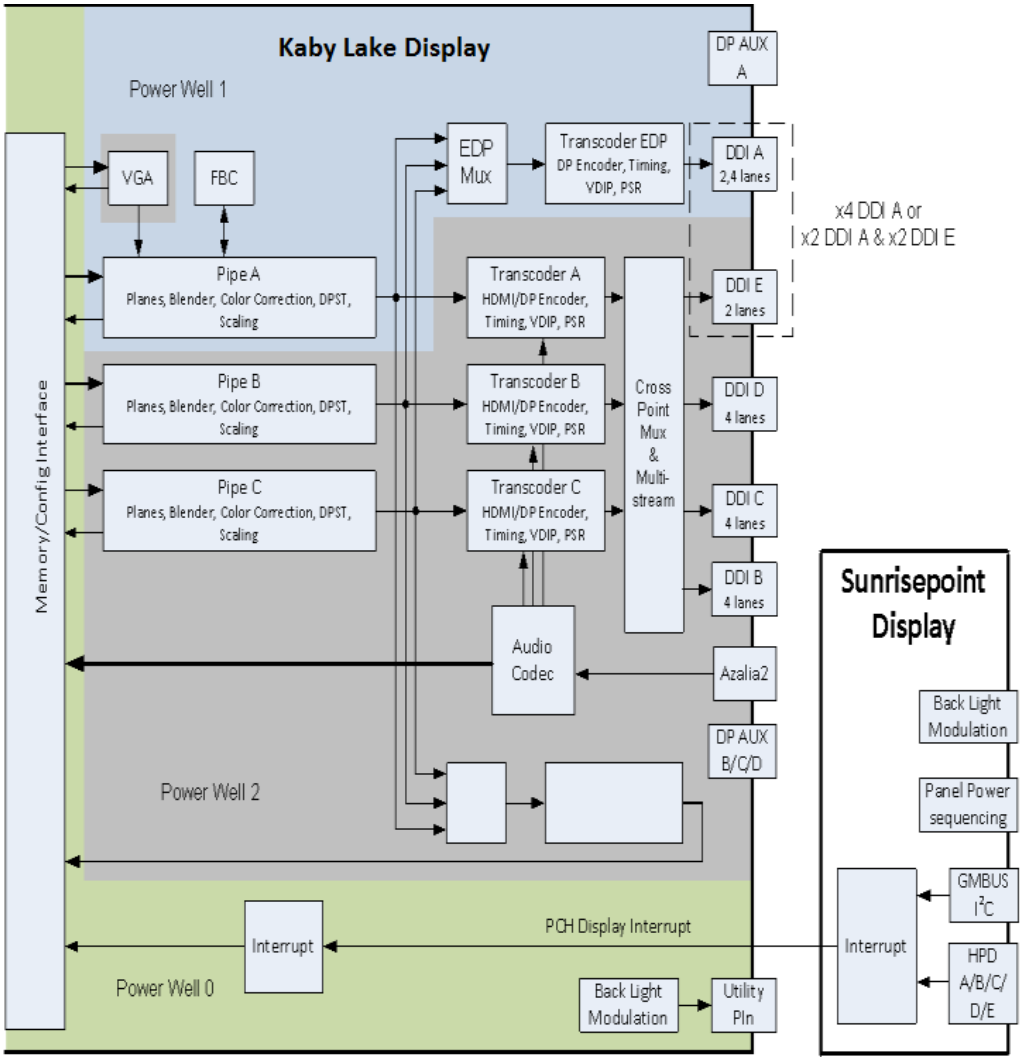
Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

This chapter contains the register descriptions for the display portion of a family of graphics devices.

These registers vary by devices within the family of devices, so special attention needs to be paid to which devices use which registers and register fields.

Different devices within the family may add, modify, or delete registers or register fields relative to another device in the same family based on the supported functions of that device.

Display Connections

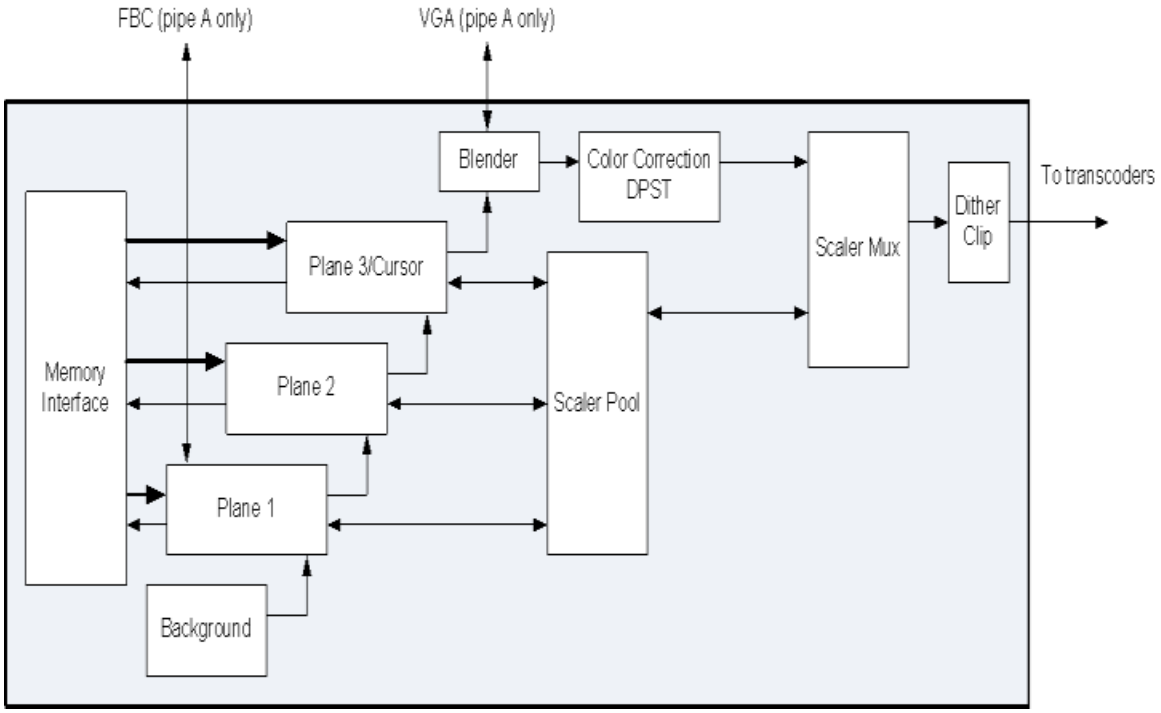


The front end of the display contains the pipes. There are three instances which are referred to as Pipe A, Pipe B, and Pipe C.

The pipes connect to the transcoders. There are five instances which are referred to as Transcoder A, Transcoder B, Transcoder C, Transcoder EDP, and Transcoder WD0.

The transcoders connect to the DDIs. There are five instances which are referred to as DDI A, DDI B, DDI C, DDI D, and DDI E.

Display Pipes



The display pipes contain the planes, blending, color correction, DPST, scaling, dithering, and clipping.

Each display pipe has three planes and a cursor. Each plane can be used as a sprite, primary, or overlay.

The background color that is seen under the bottom most plane is programmable.

The plane blending follows a fixed Z-order. Plane 1 is the bottom most plane and higher numbered planes stack on top of it.

The cursor and top most plane are mutually exclusive and cannot be both enabled at the same time.

Pipe A and Pipe B each have two pipe scalers. Pipe C has one pipe scaler.

Each pipe scaler can be assigned to scale an individual plane or scale the blended output.

Display Transcoders

The display transcoders contain the timing generators, port encoders, Audio/Video mixers, Video Data Island Packet mixers, and Panel Self Refresh controllers.

Transcoder EDP does not support HDMI, DVI, or Audio.

Transcoder WD0 only supports display capture and write back to memory.

Audio

The Azalia interface provides data to the audio codec.

The audio codec connects to the Audio/Video mixers in the transcoders.

DDIs

The DDIs contain the DisplayPort transport control and other port logic to interface to the DDI physical pins.

DDI A, DDI B, DDI C, and DDI D support lane reversal where the internal lane to package lane mapping is swapped.

DDI E does not support lane reversal.

DDI A and DDI E do not support DisplayPort multistream.

DDI A and DDI E share 2 lanes. DDI A is capable of supporting up to 4 lanes when DDI E is not connected, but only 2 lanes when DDI E is connected. DDI E is capable of supporting up to 2 lanes when connected. Dynamic switching between the two configurations is not supported.

DisplayPort A and E Lane Mapping:

Package Pin	Non-Reversed when DDI A is x4 capable	Reversed when DDI A is x4 capable	Non-Reversed when DDI A is x2 capable	Reversed when DDI A is x2 capable
DDIA/DDIE 3	Port A Main Link Lane 3	Port A Main Link Lane 0	Port E Main Link Lane 1	Port E Main Link Lane 1
DDIA/DDIE 2	Port A Main Link Lane 2	Port A Main Link Lane 1	Port E Main Link Lane 0	Port E Main Link Lane 0
DDIA/DDIE 1	Port A Main Link Lane 1	Port A Main Link Lane 2	Port A Main Link Lane 1	Port A Main Link Lane 0
DDIA/DDIE 0	Port A Main Link Lane 0	Port A Main Link Lane 3	Port A Main Link Lane 0	Port A Main Link Lane 1

DisplayPort B, C, and D Lane Mapping:

Package Pin	Non-Reversed	Reversed
DDI 3	Main Link Lane 3	Main Link Lane 0
DDI 2	Main Link Lane 2	Main Link Lane 1
DDI 1	Main Link Lane 1	Main Link Lane 2
DDI 0	Main Link Lane 0	Main Link Lane 3

HDMI/DVI TMDS Lane Mapping:

Package Pin	Non-Reversed	Reversed
DDI 3	TMDS Clock	TMDS Data2
DDI 2	TMDS Data0	TMDS Data1
DDI 1	TMDS Data1	TMDS Data0
DDI 0	TMDS Data2	TMDS Clock

DDI Equivalent Names:

DDI Name	Equivalent Names
DDI A	Port 0, DDI-A, DDIA, Port A, DDI0, eDP
DDI B	Port 1, DDI-B, DDIB, Port B, DDI1
DDI C	Port 2, DDI-C, DDIC, Port C, DDI2
DDI D	Port 3, DDI-D, DDID, Port D, DDI3
DDI E	Port 4, DDI-E, DDIE, Port E, DDI4

Pipe to Transcoder to DDI Mappings

Twin modes are not supported.

Any pipe can drive any single DDI.

With DisplayPort multistream it is possible to have multiple pipes driving a single DDI. DDI B, DDI C, and DDI D support multistream. DDI A and DDI E do not support multistream.

Pipe A can connect to either Transcoder A, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Pipe B can connect to either Transcoder B, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Pipe C can connect to either Transcoder C, Transcoder EDP, or Transcoder WD0, but not more than one simultaneously.

Transcoder A is tied to Pipe A.

Transcoder B is tied to Pipe B.

Transcoder C is tied to Pipe C.

Transcoder EDP can connect to Pipe A, Pipe B, or Pipe C, but only one at a time.

Transcoder WD0 can connect to Pipe A, Pipe B, or Pipe C, but only one at a time.

Transcoder A can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder B can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder C can connect to DDI B, DDI C, DDI D, or DDI E, but only one at a time.

Transcoder EDP can connect only to DDI A.

Transcoder WD0 does not connect to any DDI. Transcoder WD0 output only goes to memory write back.

DDI A can connect only to Transcoder EDP. DDI A does not support DisplayPort multistream.

DDI B can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI C can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI D can connect to Transcoder A, Transcoder B, or Transcoder C, individually or simultaneously if DisplayPort multistream is used.

DDI E can connect to Transcoder A, Transcoder B, or Transcoder C, individually.

Terminology

Term	Description
DP	DisplayPort
SST, DP SST	DisplayPort Single Stream Transport
MST, DP MST	DisplayPort Multi Stream Transport

Register Access Field	Description	Implementation
R/W (Read/Write)	The value written into this register will control hardware and is the same value that will be read.	Write data is stored. Read is from the stored data. Stored value is used to control hardware.
Reserved	Unused register bit. Don't assume a value for these bits. Writes have no effect.	Write data is ignored. Read is zero.
MBZ (Must Be Zero)	Always write a zero to this register.	May be implemented as Reserved or as R/W.
PBC (Preserve Bit Contents)	Software must write the original value back to this bit. This allows new features to be added using these bits.	May be implemented as Reserved or as R/W.
Read Only	The read value is determined by hardware. Writes to this bit have no effect.	Write data is ignored. Read is from a status signal or some other internal source.
Write Only	The value written into this register will control hardware. Reads return zero.	Write data is stored. Read is zero. Stored value is used to control hardware.
R/W Clear (Read/Write Clear)	Sticky status bit. Hardware will set the bit, software can clear it with a write of 1b.	Internal hardware events set a sticky bit. Read is from the sticky bit. A write of 1b clears the sticky bit.
R/W Set (Read/Write Set)	Sticky status bit. Software can set the bit with a write of 1b. Hardware will clear the bit.	A write of 1b sets a sticky bit. Internal hardware events clear a sticky bit. Read is from the sticky bit.

Register Access Field	Description	Implementation
Double Buffered	Write when desired and the written value will take effect at the time of the update point specified in the 'Double Buffer Update Point' parameter. Reads will return the written value, which is not necessarily the value being currently used to control hardware. Some have a specific arming sequence where a write to another register, specified in the 'Double Buffer Armed By' parameter, is required before the update can take place. Once the armed by register is written to, the written values of all registers controlled by that arming will take effect at the time of the double buffer update point. This is used to ensure atomic updates of several registers. Note: Once armed, by write to the armed by register, the registers controlled by this arming should not be changed until the double buffer update point is reached. If changed, this will disarm the sequence and will require another write to the armed by register to get it to the armed status again.	Two stages of registers used. Write data is stored into first stage. Read is from the first stage stored data. First stage stored value is transferred to second stage storage at the double buffer update point. Second stage stored value is used to control hardware. Arm/disarm logic may be used for some registers to control the double buffer update point.
Write/Read Status	The value written into this register will control hardware. The read value is determined by hardware.	Write data is stored. Stored value is used to control hardware. Read is from a status signal or some other internal source.

Mode Set

A mode set sequence is the programming sequence that must be followed when enabling or disabling output to a display. There are several different mode set sequences documented in the following sections. The sequence to use depends on which type of port is being enabled or disabled.

Sequences to Initialize Display

These sequences are used to initialize the display engine before any display engine functions can be enabled.

Most display engine functions will not operate while display is not initialized. Only basic PCI, I/O, and MMIO register read/write operations are supported when display is not initialized.

Initialize Sequence

1. Enable PCH Reset Handshake
    a. Set NDE_RSTWRN_OPT RST PCH Handshake En to 1b.
2. Enable Power Well 1 (PG1) and Misc IO Power
    a. Poll for FUSE_STATUS Fuse PG0 Distribution Status = 1b.
        - Timeout and fail after 5 us.
    b. Set PWR_WELL_CTL Power Well 1 Request and Misc IO Power Request to 1b.
    c. Poll for PWR_WELL_CTL Power Well 1 State and Misc IO Power State = 1b.
        - Timeout and fail after 10 us.
    d. Poll for FUSE_STATUS Fuse PG1 Distribution Status = 1b.
        - Timeout and fail after 5 us.
3. Enable CDCLK PLL
    a. Set CDCLK_CTL CD Frequency Select to the minimum.
    b. Configure DPLL0 link rate to the rate required for the eDP panel, or 810 MHz (DP 1.62 GHz) if there is no eDP panel.
    c. Set LCPLL1_CTL PLL enable to 1b.
    d. Poll for LCPLL1_CTL PLL lock = 1b.
        - Timeout and fail after 5 ms.
4. Follow Sequences for Changing CD Clock Frequency
5. Enable DBUF
    a. Set DBUF_CTL DBUF Power Request to 1b.
    b. Poll for DBUF_CTL DBUF Power State = 1b.
        - Timeout and fail after 10 us.

Un-initialize Sequence

1. Disable all display engine functions using the full mode set disable sequence on all pipes, transcoders, ports, planes, and power well 2 (PG2).
2. Disable DBUF
    a. Clear DBUF_CTL DBUF Power Request to 0b.
    b. Poll for DBUF_CTL DBUF Power State = 0b.
        - Timeout and fail after 10 us.
3. Disable CDCLK PLL
    a. Clear LCPLL1_CTL PLL enable to 0b.
    b. Poll for LCPLL1_CTL PLL lock = 0b.
        - Timeout and fail after 1 ms.
4. Disable Power Well 1 (PG1) and Misc IO Power
    a. Clear PWR_WELL_CTL Power Well 1 Request and Misc IO Power Request to 0b.
    b. Wait for 10us. Do not poll for the power well to disable. Other clients may be keeping it enabled.

Sequences for DisplayPort

This topic describes how to enable and disable DisplayPort.

Enable Sequence

Display must already be initialized

DDIA Lane Capability Control must be configured prior to enabling any ports or port clocks

1. **Enable Power Wells**

If not PipeA+DDIA - Enable Power Well 2 a. Enable PWR_WELL_CTL Power Well 2 Request b. Wait for PWR_WELL_CTL Power Well 2 State = Enabled, timeout after 20 us c. Wait for FUSE_STATUS FUSE PG2 Distribution Status = Done, timeout after 1 us

2. **If panel power sequencing is required - Enable Panel Power**
    a. Enable panel power sequencing
    b. Wait for panel power sequencing to reach the enabled state
3. **Enable Port PLL**
    a. If PLL is not already enabled, follow port clock programming sequence from Clocks section
    b. If PLL to port mapping is flexible, configure PLL to port mapping to direct the PLL output to the DDI

4. **If IO power is controlled through PWR_WELL_CTL - Enable IO Power**
    a. Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
    b. Wait for PWR_WELL_CTL DDI IO Power Request = Enabled, timeout after 20 us
5. **Enable and Train DisplayPort**
    a. Configure and enable DP_TP_CTL with link training pattern 1 selected
    b. Configure voltage swing and related IO settings. Refer to DDI Buffer section.
    c. Configure and enable DDI_BUF_CTL
    d. Gen9: Wait >518 us for buffers to enable before starting training or allow for longer time in TP1 before software timeout.
    e. Follow DisplayPort specification training sequence (see notes for failure handling)
    f. If DisplayPort multi-stream - Set DP_TP_CTL link training to Idle Pattern, wait for 5 idle patterns (DP_TP_STATUS Min_Idles_Sent) (timeout after 800 us)
    g. Set DP_TP_CTL link training to Normal.
6. **Enable Planes, Pipe, and Transcoder (repeat to add multiple pipes on a single port for multi-streaming)**
    a. If DisplayPort multi-stream - use AUX to program receiver VC Payload ID table to add stream
    b. Configure Transcoder Clock Select to direct the Port clock to the Transcoder
    c. Configure and enable planes (VGA or hires). This can be done later if desired.
    d. If VGA - Clear VGA I/O register SR01 bit 5
    e. Enable panel fitter if needed (must be enabled for VGA)
    f. Configure transcoder timings, M/N/TU/VC payload size, and other pipe and transcoder settings
    g. Configure and enable TRANS_DDI_FUNC_CTL
    h. If DisplayPort multistream - Enable pipe VC payload allocation in TRANS_DDI_FUNC_CTL
    i. If DisplayPort multistream - Wait for ACT sent status in DP_TP_STATUS and receiver DPCD (timeout after >410us)
    j. Configure and enable TRANS_CONF
    k. If panel power sequencing is required - Enable panel backlight

SRD and/or Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Notes

Changing voltage swing during link training:

- Change the swing setting following the DDI Buffer section. The port does not need to be disabled.

Changing port width (lane count) or frequency during link training:

1. Follow Disable Sequence for DisplayPort to Disable Port

2. If PLL frequency needs to change, Follow Disable Sequence for DisplayPort to Disable PLL, then follow Enable Sequence for DisplayPort to Enable PLL, using the new frequency settings
3. Follow Enable Sequence for DisplayPort to Enable and Train DisplayPort, using the new port width settings

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Enabling DisplayPort Sync Mode

See TRANS_DDI_FUNC_CTL Port Sync Mode Enable for restrictions.

1. Follow the enable sequence for the DisplayPort slave, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
    - Set TRANS_DDI_FUNC_CTL Port Sync Mode Master Select and Port Sync Mode Enable when configuring and enabling TRANS_DDI_FUNC_CTL.
2. Wait 200 uS.
3. Follow the enable sequence for the DisplayPort master, but skip the step that sets DP_TP_CTL link training to Normal (stay in Idle Pattern).
4. Set DisplayPort slave DP_TP_CTL link training to Normal.
5. Set DisplayPort master DP_TP_CTL link training to Normal.

Software may need to use DOUBLE_BUFFER_CTL to ensure updates to plane and pipe registers will take place in the same frame.

For example: If pipe A and pipe B are synchronized together and software needs the surface addresses for two planes to update at the same time, software should use DOUBLE_BUFFER_CTL when writing the surface address registers for both planes, otherwise there is a possibility that the updates could be split across a vertical blank such that one plane would update on the current vertical blank and the other plane would update on the next vertical blank.

Disable Sequence

SRD and Audio must be disabled first. Follow the audio disable sequence in the audio registers section.

1. **If panel power sequencing is required - Disable panel backlight**
2. **Disable Planes, Pipe, and Transcoder (repeat to remove multiple pipes from a single port for multi-streaming)**
    a. If VGA
        i. Set VGA I/O register SR01 bit 5 for screen off
        ii. Wait for 100 us
    b. Disable planes (VGA or hires)
    c. Disable TRANS_CONF

d. Wait for off status in TRANS_CONF, timeout after two frame times
e. If DisplayPort multistream - use AUX to program receiver VC Payload ID table to delete stream
f. If done with this VC payload
    i. Disable VC payload allocation in TRANS_DDI_FUNC_CTL
    ii. Wait for ACT sent status in DP_TP_STATUS and receiver DPCD
g. Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
h. Disable panel fitter
i. Configure Transcoder Clock Select to direct no clock to the transcoder

3. **Disable Port (all pipes and VC payloads on this port must already be disabled)**
    a. Disable DDI_BUF_CTL
    b. Disable DP_TP_CTL (do not set port to idle when disabling)
    c. Gen9: Wait 8 us or poll on DDI_BUF_CTL Idle Status for buffers to return to idle
4. **If panel power sequencing is required - Disable Panel Power**
    a. Disable panel power sequencing
5. **If IO power is controlled through PWR_WELL_CTL - Disable IO Power**
    a. Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used
6. **Disable Port PLL**
    a. If PLL to port mapping is flexible, configure PLL to port mapping to direct no clock to the DDI
    b. If this PLL is no longer needed, follow PLL disable sequence from Clocks section
7. **Disable Power Wells**
    a. If no required resource is in the power well - Disable PWR_WELL_CTL Power Well 2 Request

Disabling DisplayPort Sync Mode

1. Follow the disable sequence for the DisplayPort slave.
2. Follow the disable sequence for the DisplayPort master.

Sequences for HDMI and DVI

This topic describes how to enable and disable HDMI and DVI.

Enable Sequence

Display must already be initialized

1. **Enable Power Wells**
    a. Enable PWR_WELL_CTL Power Well 2 Request
    b. Wait for PWR_WELL_CTL Power Well 2 State = Enabled, timeout after 20 us
    c. Wait for FUSE_STATUS FUSE PG2 Distribution Status = Done, timeout after 1 us
2. **Enable Port PLL**
    a. If PLL is not already enabled, follow port clock programming sequence from Clocks section
    b. If PLL to port mapping is flexible, configure PLL to port mapping to direct the PLL output to the DDI
3. **If IO power is controlled through PWR_WELL_CTL - Enable IO Power**
    a. Enable PWR_WELL_CTL DDI IO Power Request for the DDI that will be used
    b. Wait for PWR_WELL_CTL DDI IO Power Request = Enabled, timeout after 20 us
4. **Enable Planes, Pipe, and Transcoder**
    a. Configure Transcoder Clock Select to direct the Port clock to the Transcoder
    b. Configure and enable planes (VGA or hires). This can be done later if desired.
    c. If VGA - Clear VGA I/O register SR01 bit 5
    d. Enable panel fitter if needed (must be enabled for VGA)
    e. Configure transcoder timings and other pipe and transcoder settings
    f. Configure and enable TRANS_DDI_FUNC_CTL
    g. Configure and enable TRANS_CONF
5. **Enable Port**
    a. Configure voltage swing and related IO settings. Refer to the DDI Buffer section.
    b. Configure and enable DDI_BUF_CTL

Audio can be enabled after everything is complete. Follow the audio enable sequence in the audio registers section.

Notes

If the mode set fails, follow the disable sequence to disable everything that had been enabled up to the failing point.

Disable Sequence

Audio must be disabled first. Follow the audio disable sequence in the audio registers section.

1. **Disable Planes, Pipe, and Transcoder**
    a. If VGA
        i. Set VGA I/O register SR01 bit 5 for screen off
        ii. Wait for 100 us
    b. Disable planes (VGA or hires)
    c. Disable TRANS_CONF
    d. Wait for off status in TRANS_CONF, timeout after two frame times
    e. Disable TRANS_DDI_FUNC_CTL with DDI_Select set to None
    f. Disable panel fitter
    g. Configure Transcoder Clock Select to direct no clock to the transcoder
2. **Disable Port**
    a. Disable DDI_BUF_CTL
3. **If IO power is controlled through PWR_WELL_CTL - Disable IO Power**
    a. Disable PWR_WELL_CTL DDI IO Power Request for the DDI that was used used
4. **Disable Port PLL**
    a. If PLL to port mapping is flexible, configure PLL to port mapping to direct no clock to the DDI
    b. If this PLL is no longer needed, disable it
5. **Disable Power Wells**

a. If no required resource is in the power well - Disable PWR_WELL_CTL Power Well 2 Request

Sequences for Display C5 and C6

Display C5 (DC5) is a power saving state where hardware dynamically disables power well 1 and the CDCLK PLL and saves the associated registers.

DC5 can be entered when software allows it, power well 2 is disabled, and hardware detects that all pipes are disabled or pipe A is enabled with PSR active.

Display C6 (DC6) is a deeper power saving state where hardware dynamically disables power well 0 and saves the associated registers.

DC6 can be entered when software allows it, the conditions for DC5 are met, and the PCU allows DC6.

DC6 cannot be used if the backlight is being driven from the display utility pin.

Core CPUs support DC5 and DC6.

The context save and restore program is reset on cold boot, warm reset, PCI function level reset, and hibernate/suspend.

Sequence to Allow DC5 or DC6

1. Load the correct stepping specific Display Context Save and Restore (CSR) program from the binary package.
    a. Read the package header and extract the correct individual firmware. Binary package format details can be found in sections below.
    b. Skip the header section at the start of the program binary.
    c. Copy the payload into **Display CSR Program Storage** .
    d. Perform the MMIO writes specified in the header section.
2. Configure display engine to have power well 2 disabled, following the appropriate mode set disable sequences for any ports using power well 2. This can be done earlier if desired.
3. Set display register 0x45520 bit 1 to 1b. It does not need to be cleared at any time.
4. Set DC_STATE_EN Dynamic DC State Enable = "Enable up to DC5" for DC5 or "Enable up to DC6" for DC6.

Programming Note	
Context:	Display C5 and C6
Do not switch between "Enable up to DC5" and "Enable up to DC6" without moving to "Disable" and reloading the CSR program in between. Disable DC5/DC6 during mode set and re-enable after the mode set programming is completed. MMIO accesses have more latency when DC5/DC6 is enabled. For optimal performance, disable DC5/DC6 when programming a set of registers and re-enable them after the programming is completed.	

Sequence to Disallow DC5 and DC6

1. Set DC_STATE_EN Dynamic DC State Enable = "Disable".

DMC Firmware Package

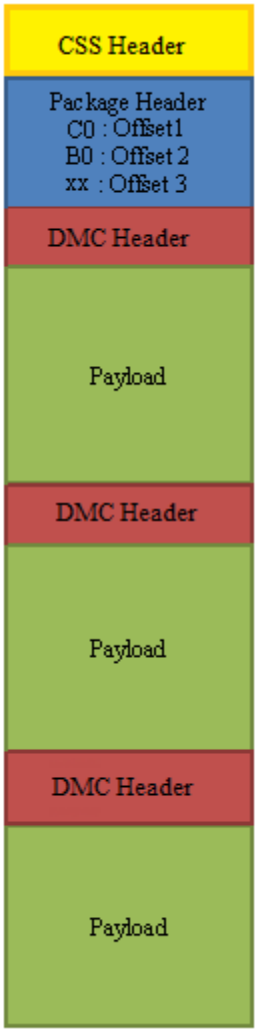
Display Micro-Controller firmware package includes all the firmwares that are required for different steppings of the product. The stepping dependent firmwares are all packaged and released as a single binary package. The package contains the CSS header, followed by the package header and the actual DMC firmwares.

Packaged firmware uses the following naming convention - <project>_dmc_ver<major>_<minor>.bin. The major version will get incremented whenever there is a change in the header layout and would require an update to the driver firmware loading module.

Major version 1

CSS Header	1.0
Package Header	1
DMC Header	1

Package Layout



CSS Header

```
typedef struct _CssHeader {
      uint32_t     moduleType;                    // 0x09 for DMC
      uint32_t     headerLen;                     // CSS header length in dwords
      uint32_t     headerVer;                     // 0x10000
      uint32_t     moduleID;                      // Not used
      uint32_t     moduleVendor;                  // Not used
      uint32_t     date;                          // YYYYMMDD(YYYY « 16 + MM « 8
+ DD)
      uint32_t     size;                          // Total dmc fw binary size in
dwords - (CSS_Headerlen + PackageHeaderLen + dmc FWsLen)/4
      uint32_t     keySize;                       // Not used
      uint32_t     modulusSize;                   // Not used
      uint32_t     exponentSize;                  // Not used
      uint32_t     reserved1[12];                 // Not used
      uint32_t     version;                       // Major Minor
      uint32_t     reserved2[8];                  // Not used
      uint32_t     uKernelHeaderInfo;             // Not used
} CssHeader;
```

Package Header

Package header contains the firmware/stepping mapping table and the corresponding firmware offsets to the individual binaries, within the package. Mapping table will list the exceptions first, followed by the default entries. An Offset value of "0xFFFFFFFF" in the mapping table indicates that there is no firmware available/supported for that stepping. The offsets to the individual binary are DWord aligned. The first individual binary starts at an offset value of "0x00000000" after the CSS Header and the Package Header.

Stepping/Version mapping example

Stepping	FW Version
A1	1.1
B*	1.6
**	2.3

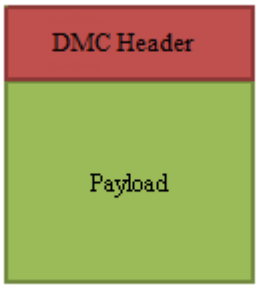
```
typedef struct _PackageHeader {
      uint8_t      headerLen;                     // DMC package header length
in dwords
      uint8_t      headerVer;                     // 0x01
```

```
      uint8_t      reserved[10];                    // Reserved
      uint32_t     numEntries;                      // Number of valid entries in
the FWInfo array below
      struct _FWInfo_ {
            uint16_t     reserved1;                 // Reserved
            char         stepping;                  // Stepping (A, B, C, ..., *).
* is a wildcard
            char         substepping;               // Sub-stepping (0, 1, ...,
*). * is a wildcard
            uint32_t     offset;                    // FW offset within the
package
            uint32_t     reserved2;                 // Reserved
      } FWInfo[20];
} PackageHeader;
```

DMC firmware binary

Each individual DMC firmware binary has a header followed by a payload whose size is specified in the header section. Along with the version, length, firmware size etc. the header section also specifies a list of MMIO addresses and data. These MMIO write cycles, in the 0x80000 - 0x8FFFF address range, should be executed as part of the initial CSR program setup.

DMC firmware Layout




```
for i = 1 to <mmioCount>
      Perform MMIO write to address <mmioaddr[i]> with data <mmiodata[i]>
typedef struct _DMCHeader {
     uint32_t    reserved;              // 0x40403E3E
     uint8_t     headerLen;             // DMC binary header length in bytes
     uint8_t     headerVer;             // 0x01
     uint16_t    dmccVer;               // Reserved
     uint32_t    project;               // Major, Minor
```

```
    uint32_t    fwSize;                 // Firmware program size (excluding header)
    in dwords
    uint32_t    fwVersion;              // Major Minor version
    uint32_t    mmioCount;              // Number of valid MMIO cycles present in
    the MMIO address and data arrays below.
    uint32_t    mmioaddr[8];            // MMIO address
    uint32_t    mmiodata[8];            // MMIO data
    unint8_t    dfile[32];              // Reserved
    uint32_t    reserved1[2];           // Reserved
} DMCHeader;
```

Display Resolution Support

A display resolution is only supported if it meets all the restrictions below for Maximum Pipe Pixel Rate, Maximum Port Link Rate, Maximum Memory Read Bandwidth, and Maximum Watermark.

Maximum Pipe Pixel Rate

The display resolution must fit within the maximum pixel rate output from the pipe. Make sure that the display pipe is able to feed pixels at a rate required to support the desired resolution.

For each enabled plane on the pipe {

If plane scaling enabled {

Horizontal down scale amount = Maximum[1, plane horizontal size / scaler horizontal window size]

Vertical down scale amount = Maximum[1, plane vertical size / scaler vertical window size]

Plane down scale amount = Horizontal down scale amount * Vertical down scale amount

Plane Ratio = 1 / Plane down scale amount

}

Else {

Plane Ratio = 1

}

If plane source pixel format is 64 bits per pixel {

Plane Ratio = Plane Ratio * 8/9

}

}

Pipe Ratio = Minimum Plane Ratio of all enabled planes on the pipe

If pipe scaling is enabled {

Horizontal down scale amount = Maximum[1, pipe horizontal source size / scaler horizontal window size]

Vertical down scale amount = Maximum[1, pipe vertical source size / scaler vertical window size]

Note: The progressive fetch - interlace display mode is equivalent to a 2.0 vertical down scale

Pipe down scale amount = Horizontal down scale amount * Vertical down scale amount

Pipe Ratio = Pipe Ratio / Pipe down scale amount

}

Pipe maximum pixel rate = CDCLK frequency * Pipe Ratio // See the display clocks section for the supported CDCLK frequencies.

Maximum Port Link Rate

The display resolution must fit within the maximum link rate for each port type.

Port Type	Maximum Link Bit Rate
eDP/DP	HBR2 5.4 GHz
HDMI	3 GHz
DVI	1.65 GHz
MIPI DSI	1.066 GHz

Maximum Memory Read Bandwidth

The display resolution must not exceed the available system memory bandwidth, considering factors like thermal throttling and bandwidth available for other memory clients.

For each pipe {

For each plane enabled on the pipe { // cursor can be ignored

Plane bandwidth MB/s = pixel rate MHz * source pixel format in bytes * plane down scale amount * pipe down scale amount

Total display bandwidth MB/s = Total display bandwidth + Plane bandwidth

}

}

Raw system memory bandwidth = (# of memory channels * memory frequency * 8 bytes)

If Total display bandwidth > system memory bandwidth available for display {Bandwidth is exceeded}

Programming Note	
Context:	Display Resolution Support
Do not use more than 60% of raw system memory bandwidth for display.	

Maximum Watermark

The display resolution must not exceed the level 0 maximum watermark value. See the volume on Watermark Programming.

Display Resolution Capabilities

These are examples of common resolutions that meet all the resolution restrictions for up to 3 simultaneous displays, 4 primary or sprite planes with 32bpp pixel format, and 1 cursor, with no panel fitter down scaling.

Attribute	CD 675 MHz	CD 540 MHz	CD 450 MHz	CD 337.5 MHz
eDP/DP x4 single stream	3840x2560 60Hz 24Bpp[4] 4096x2304 60Hz 24Bpp[4]	3840x2160 60Hz 30Bpp[4]	3200x2000 60Hz 30Bpp[3]	2880x1620 60Hz 30Bpp[2] 4096x2160 30Hz 30Bpp[2]
HDMI	Same as entry to the right	Same as entry to the right	Same as entry to the right	4Kx2K 24-30Hz 24Bpp[2]
DVI	Same as entry to the right	Same as entry to the right	Same as entry to the right	1920x1200 60Hz 24Bpp[1]

Each entry is showing the highest common resolutions at that CD clock frequency step. Lower resolutions are also supported. Higher, less common, resolutions can also work, but need to be calculated individually.

eDP, DP, DVI, and MIPI DSI are calculated using CVT 1.2 RB1 blanking and pixel rate.

HDMI is calculated using HDMI specification blanking and pixel rate.

Bpp is referring to the port output bits per pixel.

[1]Requires at least single channel DDR3 1333 for 3 simultaneous displays

[2]Requires at least single channel DDR3 1600 for 3 simultaneous displays

[3]Requires at least dual channel DDR3 1333 for 3 simultaneous displays

[4]Requires at least dual channel DDR3 1600 for 3 simultaneous displays

Examples

Example pipe pixel rate:

Plane 1 enabled at 32bpp, plane 2 enabled at 16bpp, pipe scaling enabled and down scale amount 1.12, and CDCLK 450 MHz:

Plane 1 ratio = 1

Plane 2 ratio = 1

Pipe ratio = Minimum[1, 1] = 1

Pipe ratio = 1/1.12 = 0.89

Pipe maximum pixel rate = 450 MHz * 0.89 = 400.5 MHz

Example pipe pixel rate:

Plane 1 enabled at 64bpp and plane down scale amount 1.25, plane 2 enabled at 32bpp, no panel fitting enabled, and CDCLK 540 MHz:

Plane 1 ratio = 1/1.25 * 8/9 = 0.71

Plane 2 ratio = 1

Pipe ratio = Minimum[1, 0.71] = 0.71

Pipe maximum pixel rate = 540 MHz * 0.71 = 383.4 MHz

Example memory bandwidth:

System memory bandwidth available for display = 4000 MB/s

Pipe A - Plane 1 enabled at 32bpp, plane 2 enabled at 16bpp, scaling disabled, pixel rate 148.5 MHz

Pipe B - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 148.5 MHz

Pipe C - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 148.5 MHz

Pipe A - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Pipe A - Plane 2 bandwidth = 148.5 * 2 bytes = 297 MB/s

Pipe B - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Pipe C - Plane 1 bandwidth = 148.5 * 4 bytes = 594 MB/s

Total display bandwidth = 594 + 297 + 594 + 594 = 2079 MB/s

System memory bandwidth available for display not exceeded

Example memory bandwidth:

System memory bandwidth available for display = 4000 MB/s

Pipe A - Plane 1 enabled at 32bpp, plane 2 plane enabled at 32bpp, pipe scaling enabled and down scale amount 1.12, pixel rate 414.5 MHz

Pipe B - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 414.5 MHz

Pipe C - Plane 1 enabled at 32bpp, scaling disabled, pixel rate 414.5 MHz

Pipe A - Plane 1 bandwidth = 414.5 * 4 bytes * 1.12 = 1863 MB/s

Pipe A - Plane 2 bandwidth = 414.5 * 4 bytes * 1.12 = 1863 MB/s

Pipe B - Plane 1 bandwidth = 414.5 * 4 bytes = 1658 MB/s

Pipe C - Plane 1 bandwidth = 414.5 * 4 bytes = 1658 MB/s

Total display bandwidth = 1863 + 1863 + 1658 + 1658 = 7042 MB/s

System memory bandwidth available for display **exceeded**

Clocks

CDCLK_CTL

LCPLL1_CTL

LCPLL2_CTL

WRPLL_CTL

PORT_CLK_SEL

TRANS_CLK_SEL

DPLL_CTRL1

DPLL_CTRL2

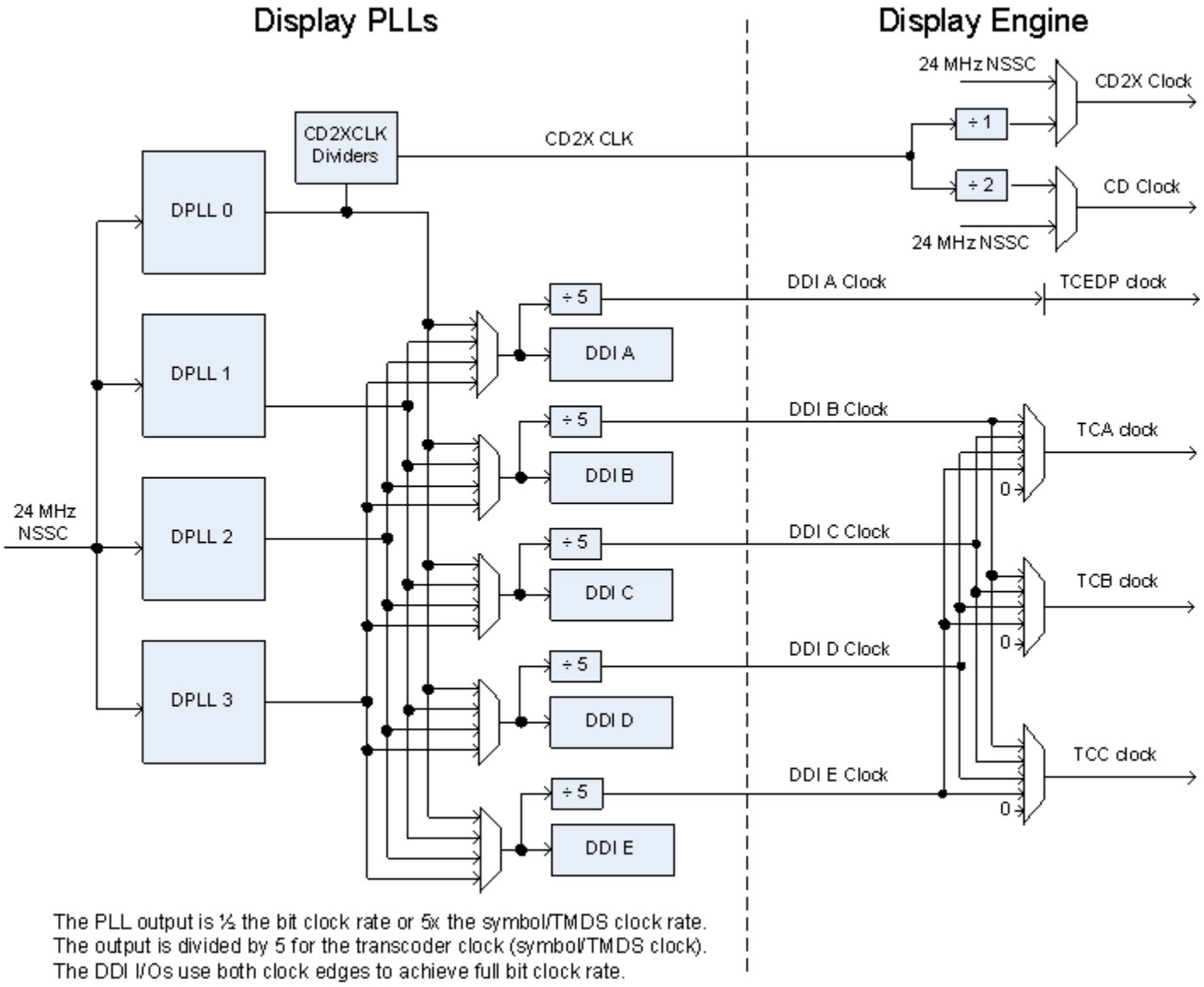
DPLL_STATUS

DPLL_CFGCR1

DPLL_CFGCR2

TIMESTAMP_CTR

Overview of Supported Display Clock Paths



The display engine clocking structure has multiple PLLs and clocks. The flow is from reference to PLL to DDI (port) clock to transcoder clock.

Display Engine Clock Reference

There is one display engine clock reference.

	Non-SSC (NSSC) Reference
Usage	Non-spread spectrum reference for DPLL and alternate source for CD and CD2X clocks
Frequency	24 MHz
Default After Reset	Enabled
Programming	Not programmable by display software.

Display Engine PLLs

There are four display engine PLLs (DPLLs).

The PLL output frequencies are 5x the symbol/TMDS rate, which is 1/2 the bit rate.

The PLL output is divided by 5 to become the symbol/TMDS clock frequency used in the display engine.

Both edges of the PLL output are used in the DDI I/Os to double the frequency to bit clock rate.

A single PLL output may be used by multiple DDI ports simultaneously if those DDI ports all require the same frequency and spread characteristics.

	DPLL0 (LCPLL 1)	DPLL1 (LCPLL 2), DPLL2 (WRPLL 1), DPLL3 (WRPLL 2)
Usage	Sources for DDI clocks and CD clock	Source for DDI clocks
Input	Non-SSC reference	Non-SSC reference
Frequency	Programmable selection in the range 0.1 to 2.7 GHz (0.2 to 5.4 GHz bit rate). CD clock usage does not allow spread spectrum to be used.	Programmable selection in the range 0.1 to 2.7 GHz (0.2 to 5.4 GHz bit rate). Spread spectrum can be added if needed.
Default After Reset	Disabled	Disabled
Programming	DPLL enable and frequency must be programmed by software when enabling and disabling a display. Frequency programming is done through the DPLL_CTRL1 register. Enable is programmed through the LCPLL1_CTL register. See the section on Port Clock Programming.	DPLL enable, SSC enable, and frequency must be programmed by software when enabling and disabling a display. Frequency and SSC programming is done through the DPLL_CTRL1 and DPLL*_CFGCR* registers. Enables are programmed through the LCPLL2_CTL and WRPLL_CTL registers. See the section on Port Clock Programming.

Recommended PLL Selection

Avoid changing frequencies on DPLL 0 since that requires DPLL 0 to be disabled, which impacts CDCLK and requires display features to be disabled.

Use DPLL 2 as a last choice because that PLL can be taken by HDPORT in some cases. See register HDPORT_STATE.

- For eDP Non-SSC, use DPLL 0 with frequency set to match the eDP panel requirement.
- For eDP SSC or external displays, use DPLLs 1, 3, 2 in that order, depending on the availability.
- If DPLL 0 is not used for eDP, run DPLL 0 in DisplayPort mode with link rate 810 MHz (DP 1.62 GHz). This will select VCO 8100 for generating CDCLK.

DDI Clocks

There is one DDI clock tied to each DDI port.

A single DDI clock output may be used by multiple transcoders simultaneously for DisplayPort Multi-streaming.

	DDI clocks A, B, C, D, and E
Usage	DDI ports I/O bit clock and symbol/TMDS clock, source for transcoder clocks
Input	Programmable selection between the DPLLs
Frequency	DPLL frequency
Default After Reset	Disabled
Programming	Mapping of DPLL to DDI must be programmed by software when enabling and disabling a display. Programming is done through the DPLL_CTRL2 registers. See the section on Port Clock Programming.

Transcoder Clocks

There is one Transcoder clock tied to each display transcoder, except Transcoder WD which uses only CD clock.

	Transcoder clock EDP	Transcoder clocks A, B, C
Usage	Transcoder EDP symbol clock	Transcoders A, B, and C symbol/TMDS clocks
Input	Always uses DDI A clock	Programmable selection between DDI clocks B, C, D, and E.
Frequency	PLL output frequency divided by 5	PLL output frequency divided by 5
Default After Reset	Connected to DDI A Clock	Disabled

Programming	Not programmable.	Mapping of DDI to Transcoder must be programmed by software when enabling and disabling a display. Programming is done through the TRANS_CLK_SEL registers.

CD Clock

CD clock refers to the Core Display clock which includes the Core Display 1X Clock (CD clock, CDclk, cdclk, CDCLK) and the Core Display 2X Clock (CD2X clock, cd2xclk, CD2XCLK).

	CD clock
Usage	Clocking for most display engine functions
Input	DPLL0 CD2X output (with several frequency options) and Non-SSC reference.
Frequency	Non-SSC reference - 24 MHz CD, 24 MHz CD2X DPLL0 with VCO 8100: • 450 MHz CD, 900 MHz CD2X • 540 MHz CD, 1080 MHz CD2X • 337.5 MHz CD, 675 MHz CD2X • 675 MHz CD, 1350 MHz CD2X DPLL0 with VCO 8640: • 432 MHz CD, 864 MHz CD2X • 540 MHz CD, 1080 MHz CD2X • 308.57 MHz CD, 617.14 MHz CD2X • 617.14 MHz CD, 1234.28 MHz CD2X
Default After Reset	Enabled with the Non-SSC reference - 24 MHz CD, 24 MHz CD2X
Programming	CD clock cannot be disabled. CD clock frequency must be programmed by software when enabling and disabling a display. Programming is done through the CDCLK_CTL register.

Port Clock Programming

DisplayPort Programming

DisplayPort PLL Enable Sequence

1. Configure DPLL_CTRL1 to enable programming, set SSC enable/disable (no SSC for DPLL0), select DP mode, and set link rate.
2. Read back DPLL_CTRL1 to ensure writes completed before the next step.
3. Enable DPLL through LCPLL1_CTL (DPLL0), LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).
4. Wait for PLL lock status in DPLL_STATUS.
5. For each DDI that will use this DPLL. Configure DPLL_CTRL2 to turn on the clock for the DDI and map the DPLL to the DDI.

DisplayPort PLL Disable Sequence

1. For each DDI that was using this DPLL. Configure DPLL_CTRL2 to turn off the clock for the DDI.
2. Disable DPLL through LCPLL1_CTL (DPLL0), LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).

Example of DisplayPort on DDIA using HBR 2.7 GHz link rate with SSC

DPLL0 does not support SSC, so this case must use one of the other DPLLs. This example assumes DPLL1 is available.

1. Configure DPLL1 in the DPLL_CTRL1 register
   - DPLL1 Override = 1b (Enable)
   - DPLL1 Link Rate = 001b (1350 MHz - DP 2.7 GHz)
   - DPLL1 SSC = 1b (Enable)
   - DPLL1 HDMI Mode = 0b (DP mode)
2. Read back DPLL_CTRL1 to ensure writes completed
3. Enable DPLL1 in the LCPLL2_CTL register
   - PLL Enable = 1b (Enabled)
4. Wait for PLL lock status in DPLL_STATUS
5. Configure DPLL1 mapping to DDIA in the DPLL_CTRL2 register
   - DDIA Select Override = 1b (Enable)
   - DDIA Clock Select = 01b (DPLL1)
   - DDIA Clock Off = 0b (On)

HDMI and DVI Programming

HDMI and DVI PLL Enable Sequence

1. Configure DPLL_CTRL1 to enable programming, disable SSC, and use HDMI mode.
2. Configure DPLL_CFGCR1 to enable programming and set DCO frequency.
3. Configure DPLL_CFGCR2 to set the dividers and DCO central frequency.
4. Read back DPLL_CTRL1, DPLL_CFGCR1, and DPLL_CFGCR2 ensure writes completed before the next step.
5. Enable DPLL through LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).
6. Wait for PLL lock status in DPLL_STATUS.
7. For each DDI that will use this DPLL. Configure DPLL_CTRL2 to turn on the clock for the DDI and map the DPLL to the DDI.

HDMI and DVI PLL Disable Sequence

1. For each DDI that was using this DPLL. Configure DPLL_CTRL2 to turn off the clock for the DDI.
2. Disable DPLL through LCPLL2_CTL (DPLL1), WRPLL_CTL1 (DPLL2), or WRPL_CTL2 (DPLL3).

Formula for HDMI and DVI DPLL Programming

Reference frequency = 24 MHz.

Symbol clock frequency MHz = DCO Frequency / (P0 * P1 * P2) / 5.

AFE clock = Symbol clock frequency MHz * 5 = DCO Frequency / (P0 * P1 * P2).

P2 can be 1, 2, 3, or 5.

If P2 != 2, then P1 must be 1. Else P1 can be 1 to 255.

If P1 == 1, then P0 can be 1, 2, 3, or 7. Else P0 can be 2, 3, or 7.

An even value for the divider (P0 * P1 * P2) is preferred.

DCO central frequency must be 8400, 9000, or 9600 MHz.

DCO frequency must be within +1% or -6% of DCO central frequency. It is preferred to get as close to the DCO central frequency as possible, but using an even divider value takes precedence.

Note: If the desired symbol clock frequency cannot be achieved with the valid values of P0, P1, P2, and DCO frequencies, use a different screen resolution with a different symbol clock frequency.

Algorithm to Find HDMI and DVI DPLL Programming

1. Find AFE clock
2. For each legal divider (P*Q*K), find the DCO. Try even dividers first. Only use odd dividers if a good even is not found.

3. Try the DCO with each central frequency.
4. Select the divider and central frequency that gives the DCO with minimum deviation from DCO central frequency and fits within the +1% and -6% requirement.
5. Find the P, Q, K values to create that divider.

Pseudo-code for HDMI and DVI DPLL Programming

```
 Minimum Positive Deviation = 1%
 Minimum Negative Deviation = 6%
 Chosen Central Frequency = NONE
 AFE Clock = 5 * Symbol Clock MHz
 Even Candidate Dividers[] = { 4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 28, 30, 32, 36, 40, 42,
44, 48, 52, 54, 56, 60, 64, 66, 68, 70, 72, 76, 78, 80, 84, 88, 90, 92, 96, 98 }
 Odd Candidate Dividers[] = { 3, 5, 7, 9, 15, 21, 35 }

 For each Divider Parity (Even, Odd) // Prefer even values for divider
    If Divider Parity == Even
       Candidate Dividers[] = Even Candidate Dividers[]
    Else
       Candidate Dividers[] = Odd Candidate Dividers[]
    For each DCO Central Frequency // Try each central frequency
       For each Candidate Dividers // Try each divider
          DCO Frequency = Candidate Divider * AFE Clock
          If DCO Frequency > DCO Central Frequency  // Positive Deviation
             DCO Central Frequency Deviation = 100 * (DCO Frequency - DCO Central Frequency)
/ DCO Central Frequency
             If (DCO Central Frequency Deviation < Minimum Positive Deviation) // Check if it
meets the minimum requirement
                Minimum Positive Deviation = DCO Central Frequency Deviation
                Chosen Central Frequency = DCO Central Frequency
                Chosen DCO Frequency = DCO Frequency
                Chosen Divider = Candidate Divider
          Else // Negative Deviation
             DCO Central Frequency Deviation = 100 * ABS(DCO Frequency - DCO Central
Frequency)  / DCO Central Frequency
             If (DCO Central Frequency Deviation < Minimum Negative Deviation)
                Minimum Negative Deviation = DCO Central Frequency Deviation
                Chosen Central Frequency = DCO Central Frequency
                Chosen DCO Frequency = DCO Frequency
                Chosen Divider = Candidate Divider
       Next Candidate Divider
    Next DCO Central Frequency
    If Chosen Central Frequency != NONE // Break out early if a good even divider is found
       break
 Next Divider Parity
 getMultiplier(num)
 {
     if (num % 2 == 0) // Even
     {
         num1 = num / 2;
         if (num1 == 1 || num1 == 2 || num1 == 3 || num1 == 5)
         {
             P0 = 2;
             P1 = 1;
             P2 = num1;
         }
         else if (num1 % 2 == 0) // Div by 4
         {
             P0 = 2;
```

```
            P1 = num1 / 2;
            P2 = 2;
        }
        else if (num1 % 3 == 0) // Div by 6
        {
            P0 = 3;
            P1 = num1 / 3;
            P2 = 2;
        }
        else if (num1 % 7 == 0) // Div by 14
        {
            P0 = 7;
            P1 = num1 / 7;
            P2 = 2;
        }
    }
    // 3, 5, 7, 9, 15, 21, 35,
    else if (num == 3 || num == 9)
    {
        P0 = 3;
        P1 = 1;
        P2 = num / 3;
    }
    else if (num == 5 || num == 7)
    {
        P0 = num;
        P1 = 1;
        P2 = 1;
    }
    else if (num == 15)
    {
        P0 = 3;
        P1 = 1;
        P2 = 5;
    }
    else if (num == 21)
    {
        P0 = 7;
        P1 = 1;
        P2 = 3;
    }
    else if (num == 35)
    {
        P0 = 7;
        P1 = 1;
        P2 = 5;
    }
}
If (Chosen Central Frequency != NONE)
   getMultiplier(Chosen Divider)
   Program DPLL_CFGCR1 with Chosen DCO Frequency/24
   Program DPLL_CFGCR2 with Chosen DCO Central Frequency, P0 (P), P1 (Q), and P2 (K)
   Return Success // Valid result found
Return Fail  // No valid result found
```

Example of DVI on DDIB using 113.309 MHz symbol clock

This example assumes DPLL3 is available.

Frequency programming algorithm finds DCO central frequency = 9000 MHz, P0=2, P1=4, P2=2, DCO Frequency = 9064.72

1. Configure DPLL3 in the DPLL_CTRL1 register
    - DPLL3 Override = 1b (Enable)
    - DPLL3 SSC = 0b (Disable)
    - DPLL3 HDMI Mode = 1b (HDMI mode)
2. Configure DPLL3 in the DPLL3_CFGCR1 register

    DCO Integer = INT(9064.72/24) = 377

    DCO Fraction = INT(((9064.72/24) - DCO Integer) * 2^15) = 22,828

    - Frequency Enable = 1b (Enable)
    - DCO Fraction = 592Ch (22828)
    - DCO Integer = 179h (377)
3. Configure DPLL3 in the DPLL3_CFGCR2 register
    - Qdiv Ratio = P1 = 04h (4)
    - Qdiv Mode = 1b (Enable)
    - Kdiv = P2 = 01b (2)
    - Pdiv = P0 = 001b (2)
    - Central Frequency = 01b (9000 MHz)
4. Read back DPLL_CTL1, DPLL3_CFGCR1, and DPLL3_CFGCR2 to ensure writes completed
5. Enable DPLL3 in the WRPLL_CTL2 register
    - PLL Enable = 1b (Enabled)
6. Wait for PLL lock in DPLL_STATUS
7. Configure DPLL3 mapping to DDIB in the DPLL_CTRL2 register
    - DDIB Select Override = 1b (Enable)
    - DDIB Clock Select = 11b (DPLL3)
    - DDIB Clock Off = 0b (On)

Example of HDMI on DDIC using 296.703 MHz symbol clock

This example assumes DPLL2 is available.

Frequency programming algorithm finds DCO central frequency = 9000 MHz, P0=1, P1=3, P2=2, DCO Frequency = 8901.09

1. Configure DPLL2 in the DPLL_CTRL1 register
    - DPLL2 Override = 1b (Enable)
    - DPLL2 SSC = 0b (Disable)
    - DPLL2 HDMI Mode = 1b (HDMI mode)

2. Configure DPLL2 in the DPLL2_CFGCR1 register

   DCO Integer = INT(8901.09/24) = 370

   DCO Fraction = INT(((8901.09/24) - DCO Integer) * 2^15) = 28,794

   - Frequency Enable = 1b (Enable)
   - DCO Fraction = 707Ah (28794)
   - DCO Integer = 172h (370)
3. Configure DPLL2 in the DPLL2_CFGCR2 register
   - Qdiv Mode = 0b (Disable) // P1=1
   - Kdiv = P2 = 01b (2)
   - Pdiv = P0 = 010b (3)
   - Central Frequency = 01b (9000 MHz)
4. Read back DPLL_CTL1, DPLL2_CFGCR1, and DPLL2_CFGCR2 to ensure writes completed
5. Enable DPLL2 in the WRPLL_CTL1 register
   - PLL Enable = 1b (Enabled)
6. Wait for PLL lock status in DPLL_STATUS
7. Configure DPLL2 mapping to DDIC in the DPLL_CTRL2 register
   - DDIC Select Override = 1b (Enable)
   - DDIC Clock Select = 10b (DPLL2)
   - DDIC Clock Off = 0b (On)

Sequences for Changing CD Clock Frequency

The CD clock frequency can be programmed to select between several frequencies.

Restrictions
The CD clock frequency impacts the maximum supported pixel rate and display watermark programming. The CD clock frequency must be at least twice the frequency of the Azalia BCLK. Do not select the CD clock frequencies that have been restricted by the Display CDCLK Limit fuse (register DFSM).
Sequence for Changing CD Clock Frequency
1. Disable all display engine functions using the full mode set disable sequence on all pipes, ports, and planes. • Includes Global Time Code 2. Inform power controller of upcoming frequency change a. Ensure any previous GT Driver Mailbox transaction is complete. b. Write GT Driver Mailbox Data0 (GTTMMADDR offset 0x138128) = 0x00000003. c. Write GT Driver Mailbox Data1 (GTTMMADDR offset 0x13812C) = 0x00000000. d. Write GT Driver Mailbox Interface (GTTMMADDR offset 0x138124) = 0x80000007.

Restrictions
e. Poll GT Driver Mailbox Interface for Run/Busy indication cleared (bit 31 = 0). • Timeout after 150 us. Do not change CD clock frequency if there is a timeout. f. Read GT Driver Mailbox Data0, if bit 0 is 0x1, continue, else go to step b. • If the condition in step f is not satisfied after cycling through steps b-f for 3 ms (typically <200 us), timeout and do not change CD clock frequency. 3. Change CDCLK_CTL register CD Frequency Select and CD Frequency Decimal to the desired frequency • The frequency change will complete within a few clock cycles. 4. Inform power controller of the selected frequency a. Write GT Driver Mailbox Data0 with the frequency selection. • If selecting 337.5 or 308.57 MHz CD clock, or disabling CD clock PLL, write 0x00000000. • If selecting 450 or 432 MHz CD clock, write 0x00000001. • If selecting 540 MHz CD clock, write 0x00000002. • If selecting 675 or 617.14 MHz CD clock, write 0x00000003. b. Write GT Driver Mailbox Data1 = 0x00000000. c. Write GT Driver Mailbox Interface = 0x80000007. • The power controller should complete within 100 us, but there is no need for display software to wait for that. 5. Update programming of register fields that are based on CD clock frequency Programming can be delayed to when the features are enabled. • Utility pin backlight frequency and duty cycle in the BLC_PWM_DATA register.

Resets

NDE_RSTWRN_OPT

The north and south display engines are reset by PCI Function Level Resets (FLR) and the chip level resets.

An FLR for Bus:Device:Function 0:2:0 resets the north and south display engines and audio codec and most of the related MMIO, PCI, and I/O configuration registers.

Display configuration registers that are reset by both the chip level reset and by FLR are marked as using the "soft" reset in the programming specification.

Display configuration registers that are reset only by the chip level reset and *not* by FLR are marked as using the "global" reset in the programming specification.

The south display engine runs panel power down sequencing (if configured to do so) before resetting.

Shared Functions

Fuses and Straps

FUSE_STATUS

DFSM

DSSM

DFSDO NE

Interrupts

MASTER_INT_CTL

GT Interrupt 0 Definition

GT Interrupt 1 Definition

GT Interrupt 2 Definition

GT Interrupt 3 Definition

DE Pipe Interrupt Definition

DE Port Interrupt Definition

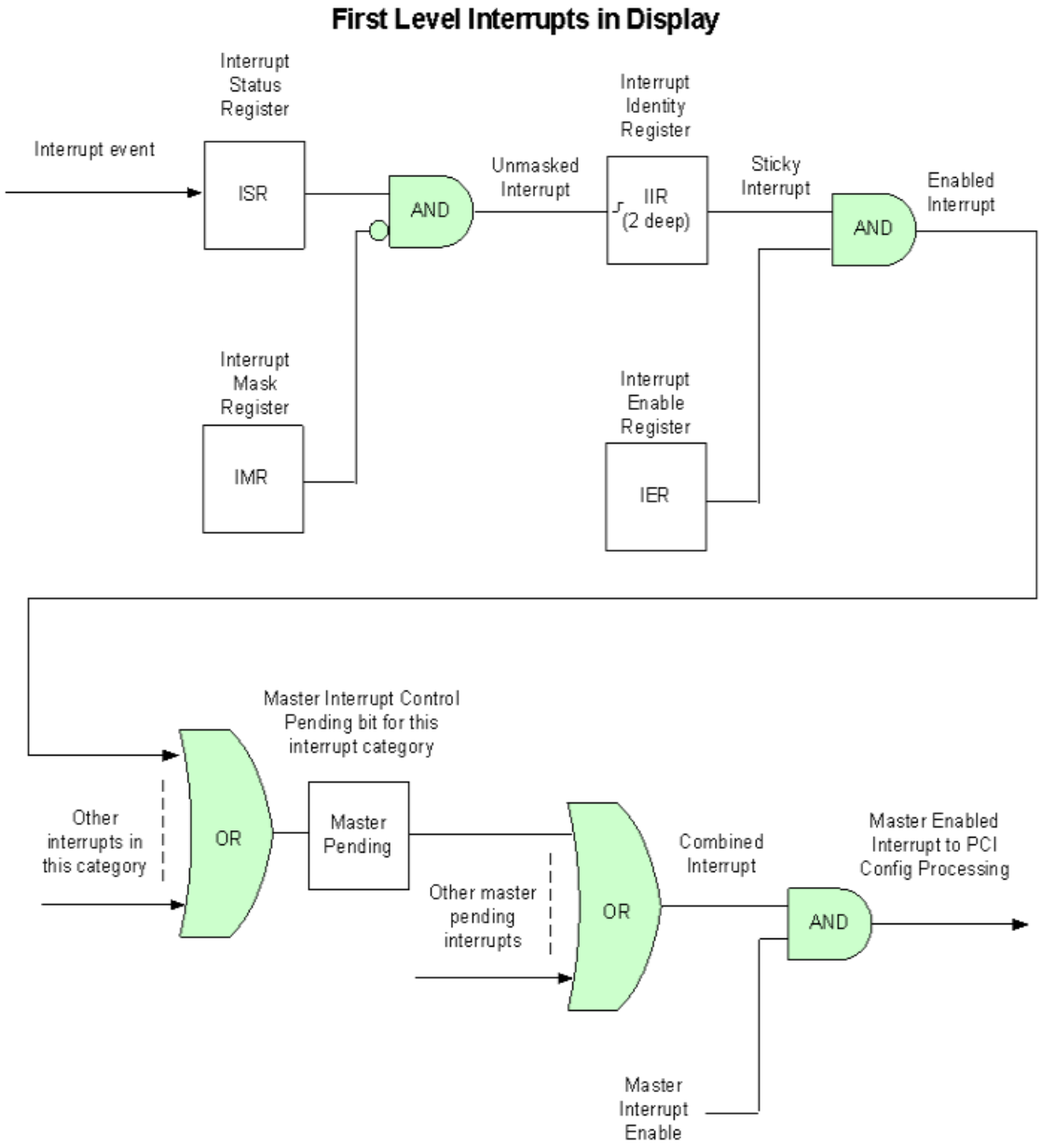
DE Misc Interrupt Definition

Audio Codec Interrupt Definition

PCU Interrupt Definition

INTERRUPT Structure

Interrupt Flow



1. For every first level interrupt bit
    a. The interrupt event comes in to the interrupt handling logic.
        - There may be more levels of interrupt handling behind each event. For example, the PCH Display interrupt event is the result of the SDE interrupt registers.
    b. The interrupt event goes to the Interrupt Status Register (ISR) where live status can be read back.
        - The live status is mainly useful for hotplug interrupts where it gives the live state of the hotplug line.
        - The live status is not useful for pulse interrupt events due to the short period that the status will be present.
    c. The interrupt event is ANDed with the inverted Interrupt Mask Register (IMR) to create the unmasked interrupt.
        - Only unmasked interrupts will proceed.
    d. The unmasked interrupt rising edge sets the sticky bit in the Interrupt Indentity Register (IIR).
        - The IIR can be cleared by writing a 1 to it.
        - The IIR can queue up to two interrupt events. When the IIR is cleared, it will set itself again if a second event was stored.
    e. The sticky interrupt is ANDed with the Interrupt Enable Register (IER) to create the enabled interrupt.
        - Only enabled interrupts will proceed.
2. All enabled interrupts are then ORed by category (Pipe, Audio Codec, Render, etc.) to create a combined interrupt which is then visible in one of the Master Interrupt Control Register (MASTER_INT_CTL) pending category bits.
3. All pending interrupts are then ORed to create the combined interrupt.
4. The combined interrupt is ANDed with the Master Interrupt Enable (MASTER_INT_CTL Bit 31) to create the master enabled interrupt.
    - Only a master enabled interrupt will proceed.
5. The master enabled interrupt then goes to PCI device 2 configuration registers PCISTS2, PCICMD2, and MC which control the MSI and line interrupt.

A Function Level Reset (FLR) or Warm Reset will reset all graphics interrupt logic, causing the master enabled interrupt to de-assert which can cause the MSI or line interrupt to de-assert.

Interrupt Service Routine

1. Disable Master Interrupt Control
    - Clear bit 31 of MASTER_INT_CTL (0x44200)
    - This is required to prevent missing any interrupts occurring back to back or during the service routine

2. Find the category of interrupt that is pending
    - Read MASTER_INT_CTL (0x44200) and record which interrupt pending category bits are set
3. Find the source(s) of the interrupt and clear the Interrupt Identity bits (IIR)
    - Read the IIR associated with each pending interrupt category, record which bits are set, then write back 1s to clear the bits that are set.
4. Process the interrupt(s) that had bits set in the IIRs
5. Re-enable Master Interrupt Control
    - Set bit 31 of MASTER_INT_CTL (0x44200)

Render Response

Display Engine Render Response Message Definition

DE_RRMR

DE_RR_DEST

Arbiter

Arbiter
ARB_CTL
ARB_CTL2

GSA

GTSP0

GTSP1 - Multiple Force Wake

GTSP2

GTSP3

GTSP4

GTSP5

GTSP6

GTSP7

Data Buffer

DBUF_CTL

DBUF_ECC_STAT

Backlight

This section refers to the CPU display backlight control. For PCH display backlight control, see South Display Engine Registers.

The backlight PWM output frequency is determined by the PWM clock frequency, increment, and frequency divider.

PWM output frequency = PWM clock frequency / PWM increment / PWM frequency divider

The frequency divider must be greater than or equal to the number of brightness levels required by software; typically 100 or 256.

Description
PWM clock frequency = CD clock frequency = 308.57 to 675 MHz PWM increment = 128 or 8, selectable by software PWM frequency divider maximum = 2^16 PWM output frequency range with PWM clock frequency 308.57 MHz and 100 brightness levels and increment 128 = 37 to 24,107 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 100 brightness levels and increment 8 = 589 to 385,713 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 256 brightness levels and increment 128 = 37 to 9,417 Hz PWM output frequency range with PWM clock frequency 308.57 MHz and 256 brightness levels and increment 8 = 589 to 150,669 Hz

Backlight Enabling Sequence

Description
1. Enable utility pin, select PWM mode, and set polarity in UTIL_PIN_CTL Util Pin Enable, Util Pin Mode, and Util Pin Output Polarity. 2. Set frequency and duty cycle in BLC_PWM_DATA Backlight Frequency and Backlight Duty Cycle. 3. Enable PWM output and set granularity in BLC_PWM_CTL PWM Enable and PWM granularity. ... 4. Change duty cycle as needed in BLC_PWM_DATA Backlight Duty Cycle.

If needed, granularity and polarity can be programmed earlier than shown.

Backlight Registers

Description
BLC_PWM_CTL
BLC_PWM_DATA

Miscellaneous Shared Functions

UTIL_PIN_CTL - Utility Pin Control

HDPORT_STATE

TOUCH_MSG_ADDR

DOUBLE_BUFFER_CTL

SWF

GTSCRATCH

Central Power

Frame Buffer Compression

FBC Registers

FBC_CFB_BASE

FBC_CTL

DPFC_CONTROL_SA

DPFC_CPU_FENCE_OFFSET

Workaround	
Context:	FBC
To prevent missed invalidations around the time FBC is being enabled, FBC render tracking must use the "Render Tracking With Nuke" method.	

FBC Overview

Frame Buffer Compression (FBC) gives a lossless compression of the display frame buffer to save power by reducing system memory read bandwidth and increasing the time between display engine reads to system memory.

FBC is only available on specific plane(s), depending on project. See FBC_CTL for details. FBC compresses pixels for the plane(s) it is attached to as they are displayed. The compressed data is written into the Compressed Frame Buffer (CFB) in graphics data stolen memory. The compressed data is then read the next time the same line needs to be displayed. Changes to the display front buffer (currently displayed

memory surface) through host aperture (GMADR) tiling fences (on projects that support that), render (RCS), and blitter (BCS) are tracked and cause the compressed lines to be invalidated and recompressed. Flips or changes to plane size and panning cause the entire buffer to be recompressed (nuke).

FBC Compression Limit

The FBC compression limit reduces the size of the Compressed Frame Buffer (CFB) by limiting which lines will be compressed. This is used when the graphics stolen memory available for the FBC CFB is smaller than the size of the original uncompressed frame buffer. There is also a FBC compressed vertical limit, listed in FBC_CTL, which is the maximum number of lines FBC can compress. Lines beyond the vertical limit do not need to be accounted for in the CFB size.

When the compression limit is 1:1, every line is written to the CFB, so the CFB width is the same as the original uncompressed frame buffer.

When the compression limit is 2:1, only lines that compress to 1/2 their original size will be written to the CFB, so the CFB width can be 1/2 the original uncompressed frame buffer.

When the compression limit is 4:1, only lines that compress to 1/4 their original size will be written to the CFB, so the CFB width can be 1/4 the original uncompressed frame buffer.

CFB size = ((Stride of plane uncompressed surface / FBC compression limit) * MIN(FBC compressed vertical limit, plane vertical source size)

FBC Programming Overview

1. Set up the compressed frame buffer.
    - The compressed buffer resides in graphics data stolen memory.
    - The stolen memory must be contiguous and un-cached.
    - The stolen memory needed for compressed frame buffer must be greater or equal to CFB size (calculation above).
    - Manage the compressed buffer size at run-time by balancing other graphics memory needs with the FBC allocation, and implement appropriate memory needs prioritization schemes.
2. Tracking for CPU host front buffer modifications through aperture fences (on projects that support that)
    - Setup System Agent (SA) registers to track CPU modifications on the display front buffer.
3. Tracking for display front buffer rendering
    - Setup Target Base Address, Front Buffer Target, and Address Valid for FBC
    - Enable Address Valid before rendering to the front buffer.
    - If needed, send nuke LRIs after each render submission to the display front buffer
4. Tracking from display front buffer BLTs
    - Setup Target Base Address and Address Valid for FBC
    - Enable Address Valid when BLTs target the display front buffer and disable when BLTs do not target the display front buffer

- Send nuke LRI or cache clean LRI after each blitter submission to the display front buffer

LRI commands to display address 0x50380 are used as part of the render and blitter tracking. Those LRIs must be followed SRM commands to the same address.

LRI to address 0x50380 with data 0x00000004 tells FBC to nuke and invalidate the entire compressed buffer.

LRI to address 0x50380 with data 0x00000002 tells FBC that previous blitter submissions have been flushed to memory and the invalidate lines can now be recompressed.

Render and blitter tracking can invalidate individual lines or nuke the entire buffer. The nuke method is used for workarounds.

Render Tracking With Nuke

- Software must send the nuke LRI after each render to the display front buffer

1. Render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
2. More render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
3. Render commands that do not touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL

Render Tracking Without Nuke

- Render tracking must be setup by software, then hardware will track FBC line invalidations automatically.

1. Render commands that potentially touch the display front buffer
    a. LRIs to FBC_RT_BASE_ADDR_REGISTER_<UPPER,LOWER> with display front buffer address, front buffer target=1, base address valid for FBC=1
    b. PIPE_CONTROL to ensure LRI takes effect
    c. Render submission
    d. PIPE_CONTROL

2. More render commands that touch the display front buffer
    a. Render submission
    b. PIPE_CONTROL
3. Render commands that do not touch the display front buffer
    a. Optional: LRIs to FBC_RT_BASE_ADDR_REGISTER_<UPPER,LOWER> with base address valid for FBC=0. Optional because hardware will detect that the render is not to the display front buffer address.
    b. Optional: PIPE_CONTROL to ensure LRI takes effect
    c. Render submission
    d. PIPE_CONTROL

Blitter Tracking With Nuke

- Software must send the nuke LRI after each BLT to the display front buffer. Never set 221d0h bit 3 (Address Valid for FBC).

1. BLT commands that touch the display front buffer
    a. BLT submission
    b. MI_FLUSH_DW
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
2. More BLT commands that touch the display front buffer (can leave Address Valid set from previous)
    a. BLT submission
    b. MI_FLUSH_DW
    c. LRI to address 0x50380 with data 0x00000004 (nuke)
    d. SRM to read address 0x50380 and store to a scratch page
3. BLT commands that do not touch the display front buffer
    a. BLT submission
    b. MI_FLUSH_DW

Blitter Tracking Without Nuke

- Blitter tracking must be enabled by software only when BLTs touch the display front buffer. Software must send the cache clean LRI after each BLT to the display front buffer.

1. BLT commands that definitely touch the front buffer
    a. LRI to address 221d0h to set bit 3 (Address Valid for FBC)
    b. MI_FLUSH_DW to ensure LRI takes effect
    c. BLT submission

    d. MI_FLUSH_DW
    e. LRI to address 0x50380 with data 0x00000002 (cache clean)
    f. SRM to read address 0x50380 and store to a scratch page
2. More BLT commands that touch the front buffer (can leave Address Valid for FBC set from previous)
    a. BLT submission
    b. MI_FLUSH_DW
    c. LRI to address 0x50380 with data 0x00000002 (cache clean)
    d. SRM to read address 0x50380 and store to a scratch page
3. BLT commands that do not touch the front buffer (must clear Address Valid for FBC)
    a. LRI to address 221d0h to set bit 3 (Address Valid for FBC)
    b. MI_FLUSH_DW to ensure LRI takes effect
    c. BLT submission
    d. MI_FLUSH_DW

CPU Host Aperture Tracking

- Host tracking must be setup by software, then hardware will track FBC line invalidations to the specified tiling fence.

1. Program Y offset from the CPU fence to the Display Buffer base in DPFC_CPU_FENCE_OFFSET.
2. Program CPU fence ID and enable CPU fence tracking in DPFC_CONTROL_SA.

Disable CPU Fence tracking in DPFC_CONTROL_SA if no tiling fence is mapped to display buffer.

Display Plane Enabling with FBC

- FBC has to be enabled after the attached display plane is enabled.
- This is the general sequence. See FBC_CTL for any workarounds.

1. Enable display plane
2. Enable FBC as described in FBC_CTL register

Display Plane Disabling with FBC

- FBC has to be disabled prior to FBC attached display plane disabling.
- This is the general sequence. See FBC_CTL for any workarounds.

1. Disable FBC as described in FBC_CTL register
2. Disable display plane.

Watermarks

The watermark registers are used to control the display to memory request timing. The watermarks must be programmed according to the Display Watermark Programming section.

Description or Link
WM_PIPE, WM_LP, and WM_LP_SPR have been replaced by PLANE_WM and moved into the planes section.
WM_MISC **WM_LINETIME**

DC States

DC_STATE_EN

Power Wells

When a power well is disabled (powered down), access to any registers in the power well will complete, but write data will be dropped and read data will be all zeroes.

The power well enable requests from all sources are logically ORd together to enable a power well, so the power well will only disable after all sources have requested the power well to disable.

Register Link
PWR_WELL_CTL

Pipe

Color Space Conversion

CSC_COEFF

CSC COEFFICIENT FORMAT

CSC_PREOFF

CSC_POSTOFF

CSC_MODE

The high color channel is the most significant bits of the color. The low color channel is the least significant bits of the color. The medium color channel is the bits between high and low. For example: In RGB modes Red is in the High channel, Green in Medium, and Blue in Low. In YUV modes, V is in the High channel, Y in Medium, and U in Low.

The color space conversion registers are double buffered and are updated on the start of vertical blank following a write to the CSC Mode register for the respective pipe.

The matrix equations are as follows:

OutputHigh = (CoefficientRY * InputHigh) + (CoefficientGY * InputMedium) + (CoefficientBY * InputLow)

OutputMedium = (CoefficientRU * InputHigh) + (CoefficientGU * InputMedium) + (CoefficientBU * InputLow)

OutputLow = (CoefficientRV * InputHigh) + (CoefficientGV * InputMedium) + (CoefficientBV * InputLow)

Example programming for RGB to YUV is in the following table:

The input is RGB on high, medium, and low channels respectively and the desired YUV output is VYU on high, medium, and low channels respectively.

Program CSC_MODE to put gamma before CSC.

Program the CSC Post-Offsets to +1/2, +1/16, and +1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

	Bt.601		Bt.709	
	Value	Program	Value	Program
RU	0.2990	0x1990	0.21260	0x2D98
GU	0.5870	0x0968	0.71520	0x0B70
BU	0.1140	0x3E98	0.07220	0x3940
RV	-0.1687	0xAAC8	-0.11460	0xBEA8
GV	-0.3313	0x9A98	-0.38540	0x9C58
BV	0.5000	0x0800	0.50000	0x0800
RY	0.5000	0x0800	0.50000	0x0800
GY	-0.4187	0x9D68	-0.45420	0x9E88
BY	-0.0813	0xBA68	-0.04580	0xB5E0

Example programming for YUV to RGB is in the following table:

The input is VYU on high, medium, and low channels respectively.

The output is RGB on high, medium, and low channels respectively.

Program CSC_MODE to put gamma after CSC.

Program the CSC Pre-Offsets to -1/2, -1/16, and -1/2 for high, medium, and low channels respectively.

The coefficients and pre and post offsets can be scaled if desired.

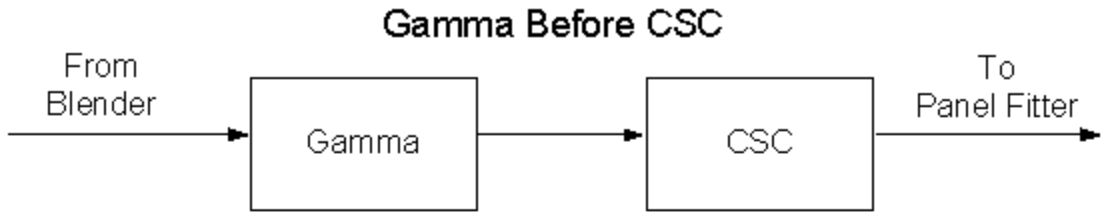
	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GY	1.000	0x7800	1.000	0x7800
BY	0.000	0x0000	0.000	0x0000
RY	1.371	0x7AF8	1.574	0x7C98
GU	1.000	0x7800	1.000	0x7800
BU	-0.336	0x9AC0	-0.187	0xABF8
RU	-0.698	0x8B28	-0.468	0x9EF8

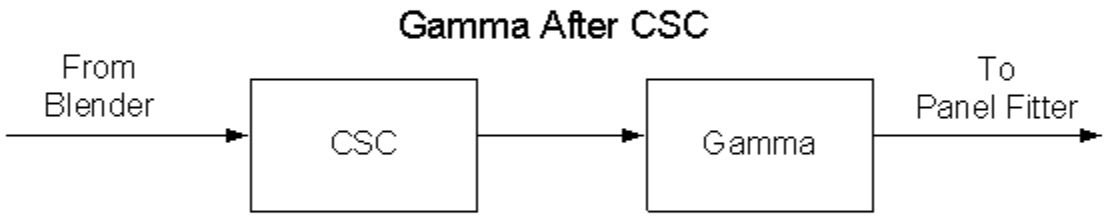
	Bt.601 Reverse		Bt.709 Reverse	
	Value	Program	Value	Program
GV	1.000	0x7800	1.000	0x7800
BV	1.732	0x7DD8	1.855	0x7ED8
RV	0.000	0x0000	0.000	0x0000

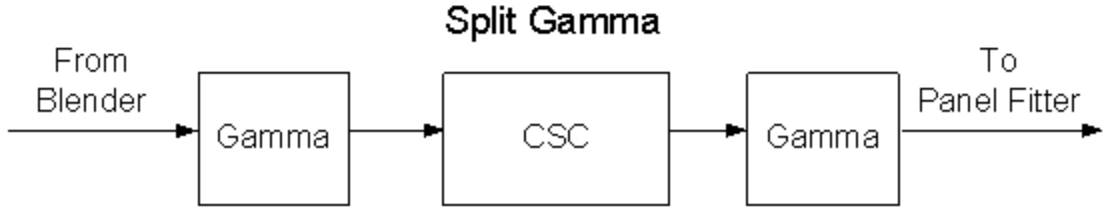
The pipe gamma and color space conversion blocks can be placed in three different arrangements:

- Gamma before CSC: Selected through the CSC Mode register. Mostly used for RGB to YUV conversion.
- Gamma after CSC: Selected through the CSC Mode register. Mostly used for YUV to RGB conversion or linear RGB to RGB conversion. Can be used with pipe color gamut enhancement.
- Split gamma: Selected through the Gamma Mode register. Mostly used for RGB to RGB conversion. Can be used with pipe color gamut enhancement. The pipe gamma enable per plane will control whether a plane will go through both gamma blocks. It is not possible to send a plane through one gamma block and not the other.

In either arrangement, the final output of the pipe gamma and CSC and gamut enhancement logic is clamped to fit in the 0 to 1.0 range before going to the ports.







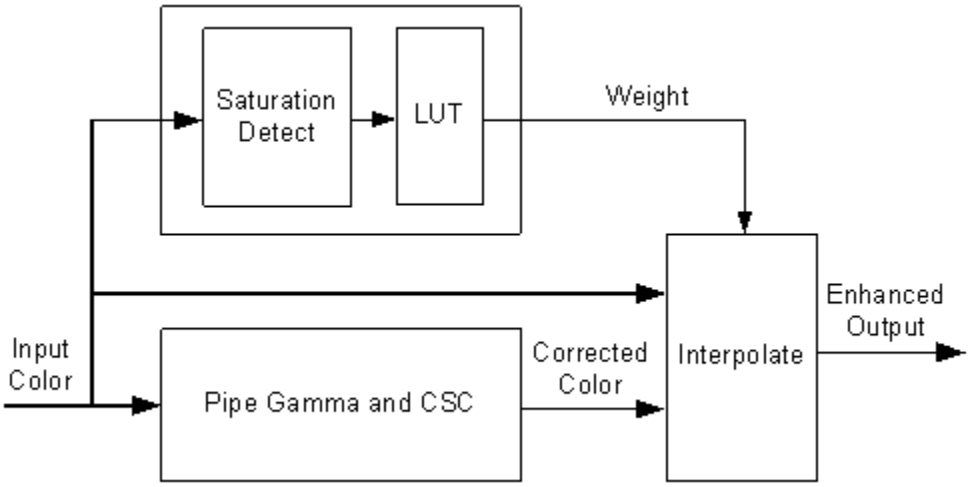
Pipe Color Gamut Enhancement

Pipe color gamut enhancement is used to enhance display of standard gamut content on wide gamut displays. It processes the color value from before and after the pipe gamma and color space correction blocks to create the color gamut enhanced output. The typical usage is to output the pipe gamma and CSC corrected color for areas of low saturated content and the input (not gamma or CSC corrected) color for areas of high saturated content. It is not recommended to use color gamut enhancement with wide gamut inputs.

CGE_CTRL

CGE_WEIGHT

The pipe Gamma and CSC must be programmed to either the split gamma mode or gamma after CSC mode when using pipe color gamut enhancement.



The saturation level of the pipe gamma and CSC input color is detected and used to index into a look up table (LUT) containing programmable weights. The saturation values are linearly distributed across the LUT indexes from the lowest index for lowest saturation to the highest index for highest saturation.

The enhanced output color is created by using the weight value to interpolate between the input color and corrected color. See the following table of weights to amount of input or corrected color used to create the enhanced output color.

Weighting of input and corrected colors

Weight from LUT	Amount of Input Color in Enhanced Output	Amount of Corrected Color in Enhanced Output
00 0000b (minimum)	0%	100%
...	...	...
00 1000b	25%	75%
...	...	...
01 0000b	50%	50%
...	...	...
01 1000b	75%	25%
...	...	...
10 0000b (maximum)	100%	0%

Example weight programming

CGE LUT Index	CGE Weight Value Decimal	CGE Weight Value Binary	CGE Weight Percent Input Color	CGE Weight Percent Corrected Color
0 (lowest saturation)	0	00 0000b	0%	100%
1	0	00 0000b	0%	100%
2	0	00 0000b	0%	100%
3	0	00 0000b	0%	100%
4	0	00 0000b	0%	100%
5	0	00 0000b	0%	100%
6	1.6	00 0010b	5%	95%
7	3.2	00 0011b	10%	90%
8	4.8	00 0101b	15%	85%
9	6.4	00 0110b	20%	80%
10	8.64	00 1001b	27%	73%
11	12.8	00 1101b	40%	60%
12	19.2	01 0011b	60%	40%
13	25.6	01 1010b	80%	20%
14	28.8	01 1101b	90%	10%
15	32	10 0000b	100%	0%
16 (highest saturation)	32	10 0000b	100%	0%

Pipe DPST

DPST_CTL

DPST_BIN

DPST_GUARD

Pipe Palette and Gamma

The display palette provides a means to correct the gamma of an image stored in a frame buffer to match the gamma of the monitor or presentation device. Additionally, the display palette provide a method for converting indexed data values to color values for VGA and 8-bpp indexed display modes. The display palette is located after the plane blender. Using the individual plane gamma enables, the blended pixels can go through or bypass the palette on a pixel by pixel basis.

PAL_LGC

PAL_PREC_INDEX

PAL_PREC_DATA

PAL_GC_MAX

PAL_EXT_GC_MAX

GAMMA_MODE

If any gamma value to be programmed exceeds the maximum allowable value in the associated gamma register, then the programmed value must be clamped to the maximum allowable value.

Programming Modes

The display palette can be accessed through multiple methods and operate in one of four different modes as follows.

8 bit legacy palette/gamma mode:

This provides a palette mode for indexed pixel data formats (VGA and primary plane 8 bpp) and gamma correction for legacy programming requirements.

All input values are clamped to the 0.0 to 1.0 range before the palette/gamma calculation. It is not recommended to use legacy palette mode with extended range formats.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the 256 palette/gamma entries. The 256 entries are stored in the legacy palette with 8 bits per color in a 0.8 format with 0 integer and 8 fractional bits.

The legacy palette is programmable through both MMIO and VGA I/O registers. Through VGA I/O, the palette can look as though there are only 6 bits per color component, depending on programming of other VGA I/O registers.

10 bit gamma mode:

This provides the highest quality gamma for pixel data formats of 30 bits per pixel or less.

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 1024 gamma entries. The first 1024 entries are stored in the precision palette with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 1024th and 1025th gamma entries to create the result value. The 1025th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Split gamma mode:

Split gamma mode is composed of two gamma functions. The first gamma is before pipe color space conversion (CSC) and the second is after CSC. This split gamma mode permits mapping to linear gamma, then color space conversion, then mapping to monitor gamma. This provides the highest quality pipe color space conversion and gamma correction for inputs with non-linear gamma.

First gamma (before CSC):

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 512 gamma entries. The first 512 entries are stored in the precision palette indexes 0 to 511 with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 512th and 513th gamma entries to create the result value. The 513th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

Second gamma (after CSC):

All input values are clamped to the 0.0 to 1.0 range before the gamma calculation.

For input values greater than or equal to 0 and less than 1.0, the input value is used to directly lookup the result value from one of the first 512 gamma entries. The first 512 entries are stored in the precision palette indexes 512 to 1023 with 10 bits per color in a 0.10 format with 0 integer and 10 fractional bits.

12 bit interpolated gamma mode:

This provides the highest quality gamma for pixel data formats greater than 30 bits per pixel.

The gamma correction curve is represented by specifying a set of gamma entry reference points spaced equally along the curve for values between -1 and 1. For extended values there is an extended gamma entry reference point at the maximum allowed input value.

All input values are clamped to the greater than -3.0 and less than 3.0 range before the gamma calculation.

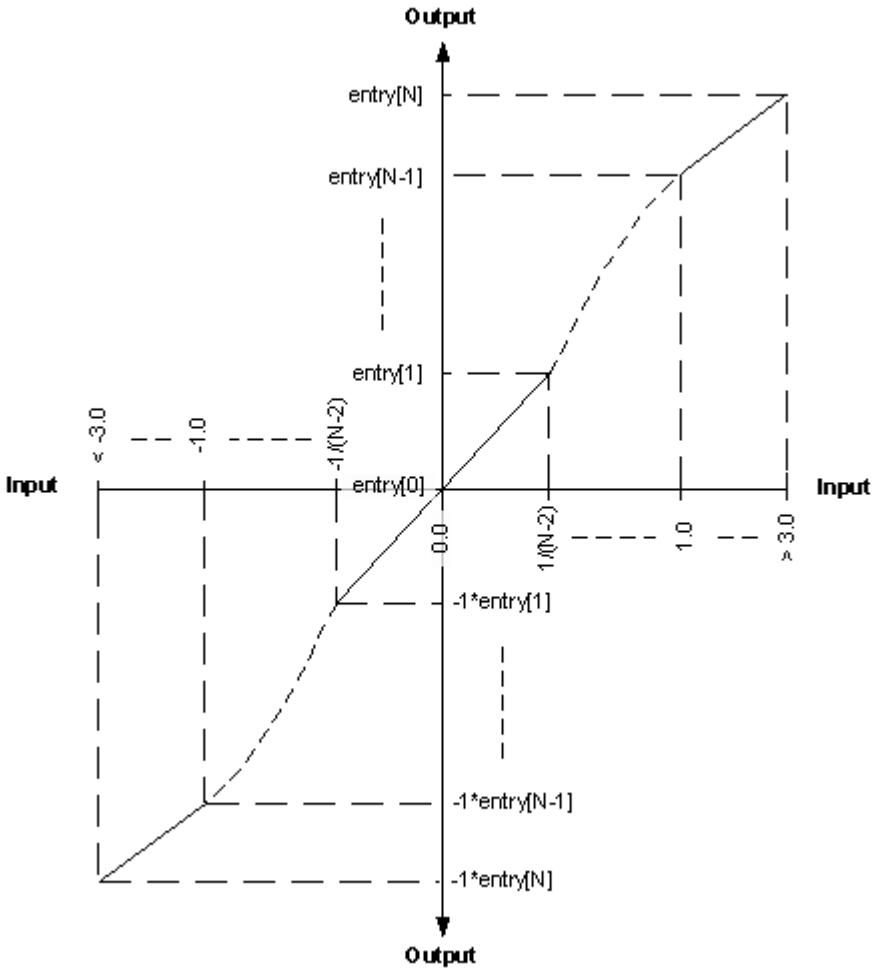
For input values greater than or equal to 0 and less than 1.0, the input value is used to linearly interpolate between two adjacent points of the first 513 gamma entries to create the result value. The first 512 entries are stored in the precision palette with 16 bits per color in a 0.16 format with 0 integer and 16 fractional bits (upper 10 bits in odd indexes, lower 6 bits in even indexes). The 513th entry is stored in the PAL_GC_MAX register with 17 bits per color in a 1.16 format with 1 integer and 16 fractional bits.

For input values greater than or equal to 1.0 and less than 3.0, the input value is used to linearly interpolate between the 513th and 514th gamma entries to create the result value. The 514th entry is stored in the PAL_EXT_GC_MAX register with 19 bits per color in a 3.16 format with 3 integer and 16 fractional bits.

For negative input values, gamma is mirrored along the X-axis, giving the same result as positive input values, except for a negative sign. When gamma input may be negative, the first gamma point should be programmed to a value of 0.0 in order to have a symmetric mirroring.

To program the gamma correction entries, calculate the desired gamma curve for inputs from 0 to 3.0. The curve must be flat or increasing, never decreasing. For inputs of 0 to 1.0, multiply the input value by 512 to find the gamma entry number, then store the desired gamma result in that entry. For inputs greater than 1.0 and less than or equal to 3.0, store the result for an input of 3.0 in the 514th gamma entry.

Example Pipe Gamma Correction Curve



Pipe Control

PIPE_SRCSZ

PIPE_SCANLINE

PIPE_SCANLINECOMP

PIPE_MISC

PIPE_BOTTOM_COLOR

PIPE_FRMCNT

PIPE_FLIPCNT

PIPE_FRMTMSTMP

PIPE_FLIPTMSTMP

Pipe Scaler

Each scaler has its own set of registers.

PS_WIN_POS

PS_WIN_SZ

PS_CTRL

PS_VSCALE

PS_HSCALE

PS_VPHASE

PS_HPHASE

PS_ECC_STAT

Planes

Plane Capability and Interoperability

Plane Assignments and Capabilities

Plane Capabilities:

	Pipe A	Pipe B	Pipe C
Plane 3	Plane or Cursor	Plane or Cursor	Plane or Cursor
Plane 2	Plane + Render Decompression	Plane + Render Decompression	Plane
Plane 1	Plane + Render Decompression, Frame Buffer compression	Plane + Render Decompression	Plane

Mapping to Command Streamer Plane Number

Display Plane Name	Command Streamer Plane Number
Plane 1 A	Plane 1
Plane 1 B	Plane 2
Plane 1 C	Plane 3
Plane 2 A	Plane 4
Plane 2 B	Plane 5
Plane 2 C	Plane 6
Plane 3 A	Plane 7
Plane 3 B	Plane 8
Plane 3 C	Plane 9
N/A	Plane 10
N/A	Plane 11
N/A	Plane 12

Plane Feature Interoperability

Display Features / Surface Formats:

	RGB32 2:10:10:10	RGB32 8:8:8:8	RGB 32-bit XR_BIAS 10:10:10	RGB64 16:16:16:16	RGB 16-bit 5:6:5	Indexed 8-bit	YUV 32-biwt 4:4:4	YUV 16-bit 4:2:2
Linear - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
Linear - 90/270 rotation								
X Tiling - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
X Tiling - 90/270 rotation								
Y Tiling (legacy) - 0/180 rotation	✓	✓	✓	✓	✓	✓	✓	✓
Y Tiling (legacy) - 90/270 rotation	✓	✓	✓				✓	✓
Yf Tiling - 0/180 rotation	✓	✓	✓		✓		✓	✓
Yf Tiling - 90/270 rotation	✓	✓	✓				✓	✓
Frame Buffer Compression		✓			✓			
Render Compression		✓						
Plane Scaling	✓	✓			✓		✓	✓
Per-pixel Alpha		✓						
Color Keying	✓	✓	✓	✓	✓	✓	✓	✓

Tiling Modes / Display Features:

	Rotation 0/180	Rotation 90/270	Render Decompression	Frame Buffer Compression	Interlacing (Interlaced Fetch)
Linear	✓			✓	✓
X Tiling	✓			✓	✓
Y Tiling Legacy	✓	✓ (some surface formats not supported)	✓	✓	
Yf Tiling	✓ (some surface formats not supported)	✓ (some surface formats not supported)	✓	✓	

Rotation Modes / Display Features:

	Interlacing	Frame Buffer Compression	Render Decompression
0/180 rotation	✓	✓	✓
90/270 rotation		✓	

Render/Display Decompression

GT Render engines uses a lossless scheme to compress the color Render targets. The goal of the compression is to reduce the memory bandwidth. The memory foot print increases slightly due to the need of the control surface.

- Decompression is supported on plane 1 and plane 2 of pipe A and pipe B with RGB8888 surface formats.
- Decompression is supported only in legacy tile Y and tile Yf surfaces.
- Decompression support is limited to left-right cache line pair mode. Top-bottom mode is not supported.
- When render decompression is enabled, display engine internally converts the Async flips to Sync flips.
- Decompression is not supported with 90/270 degree rotation.

Color Control Surface

The Color Control Surface (CCS) contains the compression status of the cache-line pairs**.** The compression state of the cache-line pair is specified by 2 bits in the CCS. Each CCS cache-line represents an area on the main surface of 16 x16 sets of 128 byte Y-tiled cache-line-pairs. CCS is always Y tiled. The

address of CCS surface is specified as an offset from the start of the Render Target main surface. CCS stride is programmed separately independent of the main surface stride.

Decompression Programming

When compressed Render targets are presented to Display, the display decompression must be enabled. Along with main surface programming, the following additional programming is required to enable the decompression

- **Decompression Enable**

Decompression is enabled by programming the 'Render Decomp' bit in the PLANE_CTL register.

- **Color Control Surface Distance**

The start of the CCS surface is programmed as the distance in bytes from the start of the main surface in the PLANE_AUX_DIST register. The CCS is always placed after the main surface and is 4K page aligned.

- **Color Control Surface Stride**

The CCS stride is programmed in the PLANE_AUX_DIST register.

Plane Rotation Programming

This topic provides programming information for plane rotation.

The 180 rotation mode is unchanged and will continue to use the same programming. In the 180 rotation mode display hardware is responsible for walking the pages in the reverse order. The cacheline walk within the page is also reversed. The 90 and 270 rotation modes require more complicated page walk mechanism. The page walk is made transparent to the hardware by providing a different set of page translations (remapped GTT) for the same rendered surface. The remapping completely abstracts the page walk away from the hardware and the hardware walks the pages as if there is no rotation. Hardware is still responsible for handling the the walk within the page appropriately. Also, the 90, 270 Rotation requires a new parameter - surface height. The changes needed in the driver programming is discussed below.

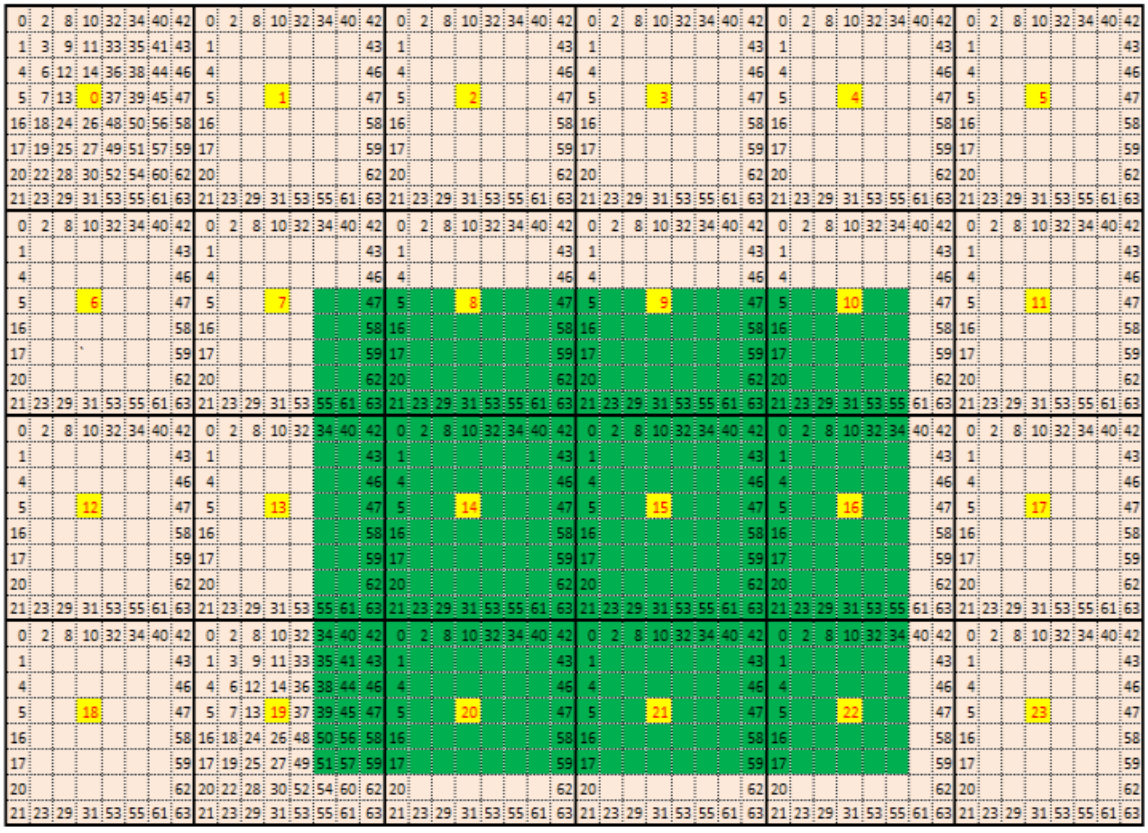
90 Rotation

For the 90 rotation programming, the plane parameters must use the following mapping

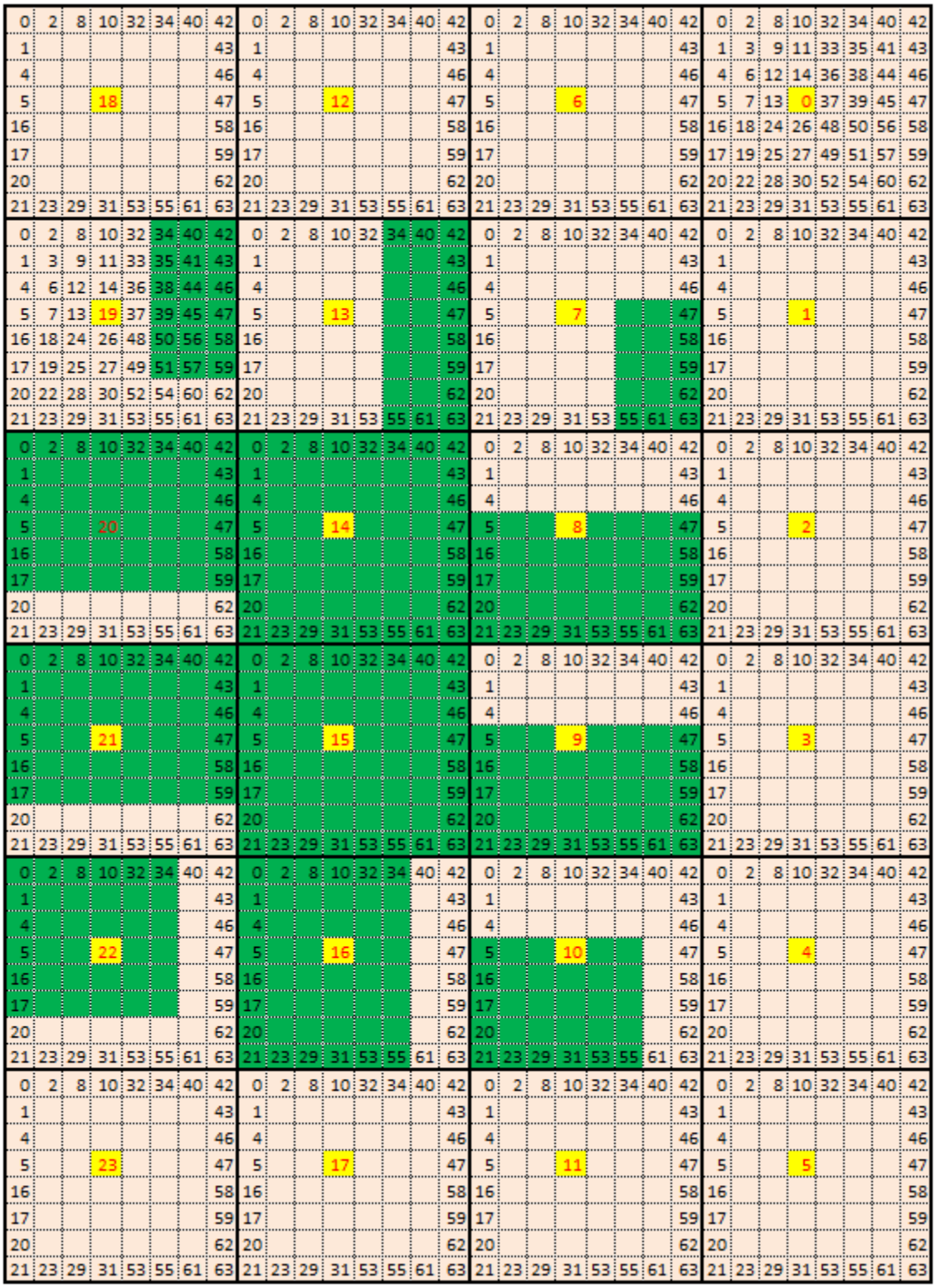
- Base address = New address (remapped GTT)
- Stride = Surface height in tiles
- Plane Width = Plane Height
- Plane Height = Plane Width
- X offset = (Surface height in tiles * tile height) - Plane Y offset - Plane Height *[Note: the calculated X offset will always be >= 0 since Plane Y Offset + Plane Height <= Surface Height in lines]*

- Y offset = X offset
- Driver can enable the Scaler, as needed, to fit the rotated content in to the pipe active display area.

Sample surface in frame buffer (Yf tiled)



Page access sequence in the rotated 90 mode



Example

Let us assume the following display programming for a single pipe – single plane, Yf tiled, non-rotated, 1920 x 1200, 4Bpp with the plane panned (100, 150), covering full active area and scaler not enabled.

GTT mapping

Here is a sample GTT mapping for 90 rotation mode.

Original GTT

Assumed Surface base = 0x200000

0x200000 = page 0

0x201000 = page 1

0x202000 = page 2

0x203000 = page 3

…

Remapped Display GTT – 90 rotation

Assumed new Surface base = 0x400000

0x400000 = page 18

0x401000 = page 12

0x402000 = page 6

0x403000 = page 0

Register programming for non-rotated scenario

- *PLANE_SURF->Surface Base Address = 0x200000*
- *PLANE_STRIDE->Stride = 60* [(1920 * 4)/128] [(width *bpp)/tile width]
- *PLANE_SIZE->Width = 1920*
- *PLANE_SIZE->Height = 1200*
- *PLANE_OFFSET->Start X Position = 100*
- *PLANE_OFFSET->Start Y Position = 150*
- *Surface Height in tiles (assumed) = 50 (allocated surface height in number of scan lines/tile height. For plane height = 1200, the surface height should be a minimum of 38 tiles (ceiling (1200/32). When panning is used, the rendered frame buffer surface will be larger than the plane size. Here, let us assume that the rendered surface height in tiles = 50).*

The programming changes to following for a 90 rotation scenario

- *PLANE_SURF->Surface Base Address = 0x400000* [uses remapped GTT]
- *PLANE_STRIDE->Stride = 50* [Surface height in tiles (assumed earlier)]
- *PLANE_SIZE->Width = 1200* [non-rotated Height]

- *PLANE_SIZE->Height* = *1920* [non-rotated Width]
- *PLANE_OFFSET->Start X Position* = *250* [(50*32)-150-1200] [(Surface height * tile height) – non rotated Y position – non rotated Height]
- *PLANE_OFFSET->Start Y Position* = *100* [non-rotated X position]

The scaler should be programmed appropriately to fit the rotated plane in the pipe active area and the window position should be adjusted if it is desired to maintain the same apparent position on a physically rotated display.

270 rotation

Uses the same GTT remapping and register programming as 90 rotation mode with the Plane control register rotation mode set as 270.

Display Buffer Programming

This topic describes display buffer allocation and shows a basic allocation method for single and multi-pipe modes. The display driver can choose to use more advanced allocation techniques as desired.

Display Buffer Allocation

Allocation of the display buffer is programmable for each display plane, using the buffer start and buffer end values in PLANE_BUF_CFG_.

Proper display buffer allocation is important for Display hardware to function correctly. Optimal allocation provides better display residency in memory low power modes. Display Buffer allocation must be recalculated and programmed when pipes/planes get enabled or disabled.

Display Buffer Size

Display Buffer Size	Total Display Buffer Blocks	Fixed Bypass Path Allocation in Blocks	Blocks Available for Driver Programming
448 KB	896	4	0 - 891

Each display buffer block is 8 cache lines.

Allocation Requirements

Allocation must not overlap between any enabled planes.

A minimum allocation is required for any enabled plane. See Minimum Allocation Requirements below.

A gap between allocation for enabled planes is allowed.

The allocation for enabled planes should be as large as possible to allow for higher watermarks and better residency in memory power saving modes.

Minimum Allocation Requirements

Allocation for each enabled plane must meet these minimum requirements.

Planes using Linear or X tiled memory formats must allocate a minimum of 8 blocks.

Planes using Y tiled memory formats must allocate blocks for a minimum number of scanlines worth of data. The formula and table of minimum scanlines is below.

Y tiled minimum allocation = Ceiling [(4 * Plane source width * Plane Bpp)/512] * MinScanLines/4 + 3

	Minimum Scanlines for Y Tile	
Plane Bpp	0/180 Rotation	90/270 Rotation
1	8	32
2	8	16
4	8	8
8	8	4

Basic Allocation Method

These are basic methods that can be used for single and multi-pipe modes. For optimal power usage, the display driver can choose to use more advanced allocation techniques as desired.

Single Pipe

Allocate a fixed number of blocks to cursors and then allocate the remaining blocks among planes, based on each plane's data rate.

BlocksAvailable = TotalBlocksAvailable

1. Allocate 32 blocks for cursor

The driver frequently enables and disables the cursor or changes the cursor pixel format. Fixed allocation is preferred for cursor to minimize the buffer re-allocation. More allocation might be required to support deeper low power states (based on the results of watermark calculations).

CursorBufAlloc = 32

BlocksAvailable = BlocksAvailable – 32

2. Check for minimum buffer requirement

For each enabled plane

If Y tiled

MinScanLines = Look up minimum scanlines needed from the table

*PlaneMinAlloc = Ceiling [(4*Plane width*Bpp)/512] * MinScanLines/4 + 3*

Else

PlaneMinAlloc = 8

If sum of PlaneMinAlloc > BlocksAvailable

Error - Display Mode can't be supported.

The driver can change the number of enabled planes or the plane configuration and rerun the algorithm.

3. Calculate Relative Data Rate for planes

In this step the driver may want to use the expected maximum plane source sizes so it does not have to reallocate for a plane that is changing size.

For each enabled plane

If PlaneScalerEnabled

*PlaneScaleFactor = (Plane width/Scaler window X size) * (Plane height/Scaler window Y size)*

Else

PlaneScaleFactor = 1

*PlaneRelativeDataRate = Plane height * Plane width * plane source bytes per pixel * PlaneScaleFactor*

4. Allocate blocks for enabled planes as per the Data rate ratio.

For each plane that needs allocation (PlaneBlockAllocFinal == false)

*PlaneBufAlloc = floor (BlocksAvailable * PlaneRelativeDataRate/Sum of PlaneRelativeDataRate of all planes that need allocation).*

*floor - rounds down to an integer value dropping the fractional part.

5. Adjust for minimum allocation requirement

AdjustmentRequired = false

For each plane needs allocation (PlaneBlockAllocFinal == false)

If PlaneBufAlloc < PlaneMinAlloc

AdjustmentRequired = true

PlaneBufAlloc = PlaneMinAlloc

PlaneBlockAllocFinal = true

BlocksAvailable = BlocksAvailable - PlaneMinAlloc

If AdjustmentRequired = true

Go back to step 4

Multi-Pipe

NumPipes = Total number of display pipes in the hardware

Allocate a fixed number of blocks to cursors, allocate *1/NumPipes* of the remaining blocks to each pipe, then calculate each pipe individually as in the Single Pipe case.

1. Allocate 8 blocks for cursor per pipe

The driver frequently enables and disables the cursor or changes the cursor pixel format. Fixed allocation is preferred for cursor to minimize the buffer re-allocation. More allocation might be required to support deeper low power states (based on the results of watermark calculations).

For each enabled cursor

CursorBufAlloc = 8

*BlocksAvailable = TotalBlocksAvailable - (8 * NumPipes)*

2. Distribute the blocks equally among the pipes

   BlocksAvailable = BlocksAvailable/NumPipes

3. Assign blocks to the planes

   For each pipe

Perform Single Pipe sequence, starting from step 2.

Buffer allocation re-distribution

When an additional pipe is getting enabled, or an existing pipe requires more buffer to support a new mode or is disabled, buffer reallocation may be necessary for proper display functionality. Whenever a portion of the allocated buffer is taken away from one pipe and allocated to a different pipe, the following sequence should be followed to make sure that there are no buffer allocation overlaps at any point of time.

1. *For each pipe whose allocation is reduced*
   a. *Program the new buffer allocation.*
   b. *Wait for VBlank of that pipe for new allocation to update.*
2. *For each pipe whose allocation is increased*
   a. *Program the new buffer allocation.*
   b. *Wait for VBlank of that pipe for new allocation to update.*

Display Buffer Allocation and Watermark programming prior to OS boot

Basic programming of the display buffer and watermarks to allow limited display usage prior to OS boot:

This will prevent package power saving states from enabling.

Supported usages:

- Up to 3 pipes enabled at once
- Up to one universal plane enabled per pipe. No cursor.
- Linear or Xtile memory
- Any RGB frame buffer pixel format 32bpp or less, without render compression
- Any screen resolution
- Downscaling less than or equal to 12.5%

Allocate 160 blocks per pipe.

Pipe A: 0-159, Pipe B: 160-319, Pipe C: 320-479

PLANE_BUF_CFG_<plane number>_A = 0x009F0000

PLANE_BUF_CFG_<plane number>_B = 0x013F00A0

PLANE_BUF_CFG_<plane number>_C = 0x01DF0140

Set level 0 watermarks for any enabled plane to 160 blocks and 2 lines.

PLANE_WM_<plane number>_<pipe>_0 = 0x800080A0

The higher level watermarks for any enabled plane must have bit 31=0 to keep the low power watermarks disabled.

VGA

The VGA Control register is located here. The VGA I/O registers are located in the VGA Registers document.

VGA_CONTROL

Cursor Plane

Planes
CUR_CTL
CUR_BASE
CUR_POS
CUR_PAL
CUR_FBC_CTL
PLANE_SURFLIVE
CUR_SURFLIVE
PLANE_BUF_CFG
PLANE_WM

The CUR_CTL and CUR_FBC_CTL active registers will be updated on the vertical blank or when pipe is disabled, after the CUR_BASE register is written, or when cursor is not yet enabled, providing an atomic update of those registers together with the CUR_BASE register.

Universal Plane

PLANE_CTL

PLANE_STRIDE

PLANE_POS

PLANE_SIZE

PLANE_SURF

PLANE_LEFT_SURF

PLANE_AUX_DIST

PLANE_OFFSET

PLANE_KEYVAL

PLANE_KEYMSK

PLANE_KEYMAX

PLANE_SURFLIVE

PLANE_WM

PLANE_BUF_CFG

PLANE_GAMC

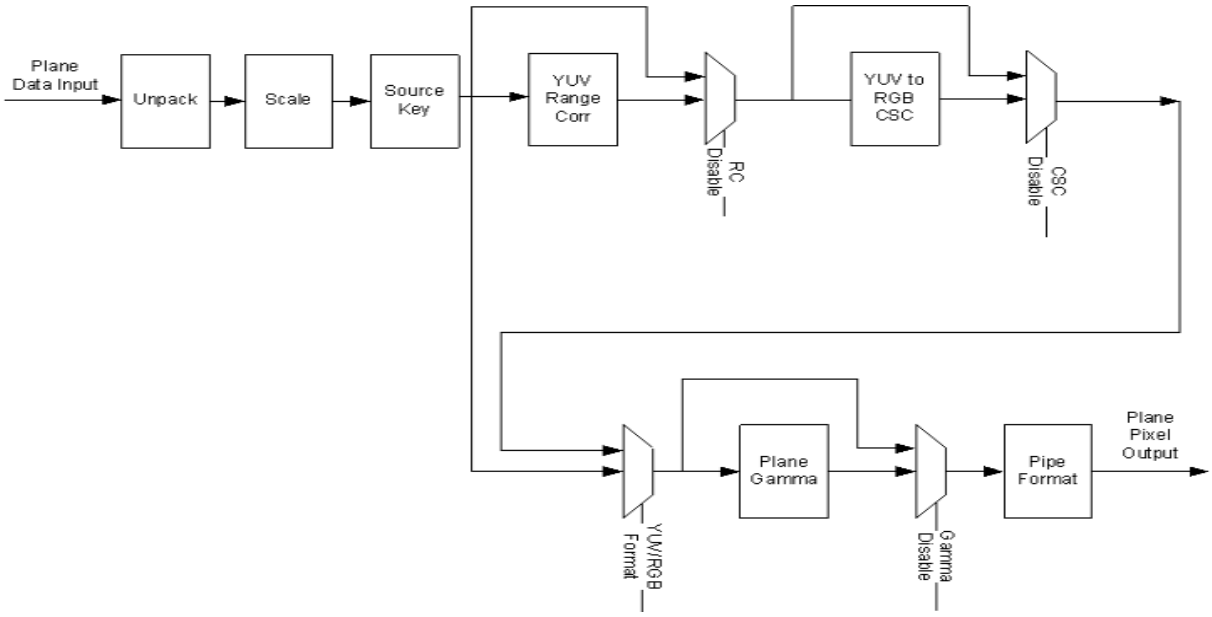
PLANE_GAMC16

PLANE_GAMC17

Many of the plane control active registers will be updated on the vertical blank or when pipe is disabled, after the surface base address register is written, or when the plane is not yet enabled, providing an atomic update of those registers together with the surface base address register.

Data flow through the plane (Steps 2-6 may be enabled or disabled by programming control bits):

1. Unpack data into pixels
2. Plane Scaling
3. Source Key
4. YUV Range Correction (can only be used by YUV source pixel formats)
5. YUV to RGB Color Space Conversion (can only be used by YUV source pixel formats)
6. Gamma Correction
7. Conversion to pipe data format



Plane Pixel Formats

Plane Source Pixel Format Mapping of Bits to Colors:

XRGB Formats	Ignored	Red	Green	Blue
RGB 32-bit 2:10:10:10 BGRX	31:30	29:20	19:10	9:0
RGB 32-bit 2:10:10:10 RGBX	31:30	9:0	19:10	29:20
RGB 32-bit 8:8:8:8 BGRX	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBX	31:24	7:0	15:8	23:16
RGB 64-bit 16:16:16:16 Float BGRX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 Float RGBX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	15:0	31:16	47:32
RGB 32-bit XR_BIAS 10:10:10	31:30	9:0	19:10	29:20
16-bit BGR 5:6:5	N/A	15:11	10:5	4:0

ARGB Formats	Alpha	Red	Green	Blue
RGB 32-bit 8:8:8:8 BGRA	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBA	31:24	7:0	15:8	23:16

XRGB Formats	Ignored	Red	Green	Blue
RGB 32-bit 2:10:10:10 BGRX	31:30	29:20	19:10	9:0
RGB 32-bit 2:10:10:10 RGBX	31:30	9:0	19:10	29:20
RGB 32-bit 8:8:8:8 BGRX	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBX	31:24	7:0	15:8	23:16
RGB 64-bit 16:16:16:16 Float BGRX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	47:32	31:16	15:0
RGB 64-bit 16:16:16:16 Float RGBX Each component is 1:5:10 MSb-sign:exponent:fraction	63:48	15:0	31:16	47:32
RGB 32-bit XR_BIAS 10:10:10	31:30	9:0	19:10	29:20
16-bit BGR 5:6:5	N/A	15:11	10:5	4:0

ARGB Formats	Alpha	Red	Green	Blue
RGB 32-bit 8:8:8:8 BGRA	31:24	23:16	15:8	7:0
RGB 32-bit 8:8:8:8 RGBA	31:24	7:0	15:8	23:16

Transcoder

Transcoder Control

TRANS_CONF

TRANS_STEREO3D_CTL

Transcoder Timing

TRANS_HTOTAL

TRANS_HBLANK

TRANS_HSYNC

TRANS_VTOTAL

TRANS_VBLANK

TRANS_VSYNC

TRANS_SPACE

TRANS_VSYNCSHIFT

TRANS_MULT

Transcoder MN Values

Description
These values are used for DisplayPort.

There is one instance of these registers per each transcoder.

For dynamic switching between multiple refresh rates, M/N values may be reprogrammed on the fly. The link N should be programmed last to trigger the update of all the data and link M and N registers and then the new M/N values will be used in the next frame that is output.

DATAM

DATAN

LINKM

LINKN

Clocks:

ls_clk is the link symbol clock. i.e. 270 MHz for HBR2.

strm_clk is the stream clock, which is the video pixel rate or dot clock.

cdclk is the core display clock.

The link only operates in Synchronous Clock mode. The link clock and stream clock are synchronous and the link M and N values stay constant for a given pixel rate.

Calculation of TU:

TU is the Transfer Unit.

TU size calculation (applies to SST only)

TU size for reduced blanking cases may vary and the following formula should be used in such cases.

TU size = MIN(INT((Horizontal Blank Width * ls_clk/strm_clk)-2), 64)

Example 1:

ls_clk = 810 MHz

strm_clk = 1025.766 MHz

Horizontal Blank = 80 pixels (Reduced Blanking v2 timings)

Calculated TU size = MIN(INT(63.1-2),64) = 61

Actual TU size = 61

So, in this example TU size becomes 61.

Example 2:

ls_clk = 810 MHz

strm_clk = 923.221 MHz

Horizontal Blank = 80 pixels (Reduced Blanking v2 timings)

TU size = MIN(INT(70.18-2),64) = 64

Calculation of Data M, and Data N:

Note that for 4:2:0 format the number of bytes per pixel will be half the number of bytes of RGB pixel.

Active/TU Size = Payload/Capacity = Data M/N

Compression Ratio (CR) = DisplayPort Compression enabled ? min(ratio1,ratio2) : 1

- ratio1 = Compressor BW / Link BW = (cdclk * bytes per pixel) / (ls_clk * number of lanes)
- ratio2 = (Horizontal active in pixels * bytes per pixel / 4) / ((Horizontal active in pixels * bytes per pixel / 8) +2)

Data M/N = (strm_clk * bytes per pixel) / (CR * ls_clk * number of lanes)

Calculation of Link M and Link N:

Link M/N = strm_clk / ls_clk

Restriction on clocks and number of lanes:

- Number of lanes >= INT(strm_clk * bytes per pixel / ls_clk)

Restrictions on the Virtual Channel (VC) payload size in DisplayPort MST mode

- In a x1 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 4.

- In a x2 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 2.
- In a x4 lane config, each pipe stream on the link must use a VC payload size that is a multiple of 1.

Transcoder Video Data Island Packet

Data Island Packet (DIP) is a mechanism that allows data to be sent over a digital port during blanking, according to the HDMI and DisplayPort specifications. This includes header, payload, checksum, and ECC information.

Each type of Video DIP will be sent once each frame while it is enabled.

VIDEO_DIP_CTL

VIDEO_DIP_DATA

VIDEO_DIP_GCP

VIDEO_DIP_ECC

Construction of DIP for AVI, VS, or SPD (HDMI only):

Dword	Byte3	Byte2	Byte1	Byte0
0	Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8 (RO)	Reserved	Reserved	Reserved	HB ECC
9 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0

HB = Header Byte, DB = Data Byte, RO = Read Only

Construction of DIP for GMP (HDMI or DisplayPort):

Dword	Byte3	Byte2	Byte1	Byte0
0	DP: HB3 HDMI: Reserved	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20

Dword	Byte3	Byte2	Byte1	Byte0
7	DB27	DB26	DB25	DB24
8 (RO)	Reserved	Reserved	Reserved	DP: Reserved HDMI: HB ECC
9 (RO)	DP: Reserved HDMI: DB ECC 3	DP: Reserved HDMI: DB ECC 2	DP: Reserved HDMI: DB ECC 1	DP: Reserved HDMI: DB ECC 0
10 (RO)	DP: HB ECC 3 HDMI: Reserved	DP: HB ECC 2 HDMI: Reserved	DP: HB ECC 1 HDMI: Reserved	DP: HB ECC 0 HDMI: Reserved
11 (RO)	DP: DB ECC 3 HDMI: Reserved	DP: DB ECC 2 HDMI: Reserved	DP: DB ECC 1 HDMI: Reserved	DP: DB ECC 0 HDMI: Reserved
12 (RO)	DP: DB ECC 7 HDMI: Reserved	DP: DB ECC 6 HDMI: Reserved	DP: DB ECC 5 HDMI: Reserved	DP: DB ECC 4 HDMI: Reserved

HB = Header Byte, DB = Data Byte, DP = DisplayPort, RO = Read Only

Construction of DIP for VSC (DisplayPort only):

Dword	Byte3	Byte2	Byte1	Byte0
0	HB3	HB2	HB1	HB0
1	DB3	DB2	DB1	DB0
2	DB7	DB6	DB5	DB4
3	DB11	DB10	DB9	DB8
4	DB15	DB14	DB13	DB12
5	DB19	DB18	DB17	DB16
6	DB23	DB22	DB21	DB20
7	DB27	DB26	DB25	DB24
8	DB31	DB30	DB29	DB28
9 (RO)	HB ECC 3	HB ECC 2	HB ECC 1	HB ECC 0
10 (RO)	DB ECC 3	DB ECC 2	DB ECC 1	DB ECC 0
11 (RO)	DB ECC 7	DB ECC 6	DB ECC 5	DB ECC 4

HB = Header Byte, DB = Data Byte, RO = Read Only

The audio subsystem is also capable of sending Data Island Packets. These packets are programmed by the audio driver and can be read by in MMIO space via the audio control state and audio HDMI widget data island registers.

Video DIP data write sequence:

1. Wait for 1 VSync to ensure completion of any pending video DIP transmissions
2. Disable the video DIP being updated (disable VDSC before updating PPS DIP)
3. Program video DIP data buffer registers for DIP being updated
4. Enable the video DIP

The video DIP data and ECC buffers may be read at any time.

DIP data buffer registers must be programmed with valid data before enabling the DIP.

Partial DIPs are never sent out while the port is enabled. Disabling the DIP at the same time it is being transferred will result in the DIP being completed before the function is disabled.

Shutting off the port on which DIP is being transmitted will result in partial transfer of DIP data. There is no need to switch off the DIP enable bit if the port transmitting DIP is disabled.

When disabling both the DIP port and DIP transmission, first disable the port and then disable DIP.

Enabling a DIP function at the same time that the DIP would have been sent out (had it already been enabled) will result in the DIP being sent on the following frame.

For HDMI, even if no DIP is enabled, a single Null DIP will be sent at the same point in the stream that DIP packets would have been sent.

Transcoder DDI Function

TRANS_DDI_FUNC_CTL

TRANS_MSA_MISC

Panel Self Refresh

This section is about Panel Self Refresh (PSR). PSR1 at one point was named Self Refreshing Display (SRD).

PSR1 enable sequence:

- Prerequisite: The associated transcoder and port are running.

1. Configure FBC host and render tracking. The FBC function does not need to be enabled in FBC_CTL.
2. Program Transcoder EDP VSC DIP data with a valid setting for SRD/PSR.
3. Configure and enable SRD_CTL.

PSR1 disable sequence:

- Prerequisite: The associated transcoder and port are running.

1. Disable SRD_CTL.
2. Wait for SRD_STATUS to show SRD is Idle. This will take up to one full frame time (1/refresh rate), plus SRD exit training time (max of 6ms), plus SRD aux channel handshake (max of 1.5ms).

PSR2 disable sequence:

1. Program PSR2_CTL reset Psr2Enable, SelectiveUpdateTrackingEnable bits.
2. Disable GTC if required.
3. Wait for PSR2_STATUS to show PSR2 is Idle. This will take up to one full frame time (1/refresh rate), plus exit training time (max of 6ms), plus aux channel handshake (max of 1.5ms).

Registers
SRD_CTL
SRD_STATUS
SRD_PERF_CNT
SRD Interrupt Bit Definition
SRD_IMR
SRD_IIR
PSR_MASK
PSR_EVENT
PSR2_CTL
PSR2_MAN_TRK_CTL
PSR2_SU_STATUS

Registers
PSR2_STATUS

Transcoder Port Sync

Feature Description

PORT SYNC is a transcoder level feature. This mode forces two or more transcoders to be in sync with one transcoder master and one or more transcoder slaves. In the case of DP/eDP, the master is unaware that it is operating in Port Sync mode. Only the slave is aware that it is operating in this mode. Hence, port sync mode is only enabled in the slave transcoder.

DP/eDP Port Sync Restrictions

1. The slave and master transcoders and associated ports must have identical parameters and properties.
2. They must be connected to the same PLL, have the same color format, link width (number of lanes enabled), resolution, refresh rate, dot clock, TU size, M and N programming, etc.
3. Port Sync Mode must only be enabled with DisplayPort SST.
4. PSR would need to be disabled when port sync mode is enabled.
5. Port Sync Mode Master Select must be programmed with a valid value when Port sync Mode is enabled.

Audio

This section describes Audio features.

Audio Bios Programming Sequence

Codec Verb Table

For each codec present on the High Definition Audio codec link, a corresponding pre-defined “Codec Verb Table” must be available to System BIOS. The Codec Verb Tables are based on codec specific information (coded datasheet) and platform design specific information (schematics) and are built by System BIOS writers and platform designers. The table contains a list of 32-bit “Verb”s (command and data payload) to be sent to the corresponding codec over the High Definition Audio codec link.

Below is a sample High Definition Audio Codec Verb Table for a platform with 1 codec at codec address 01h.

;Sample HIGH DEFINITION AUDIO Codec Verb Table

;Codec Address (CAd) = 02h

;Codec Vendor: XYZ Company

;VenID DevID:

```
    dd 12345678h
;-----------------------------------------------------
; FrontPanel_Supported?   ; 1=Supported ,0=Not supported
    db  01h
; # of Rear Panel Pin Complexes
   dw 000Ch
; # of Front Panel Pin Complexes
dw 0002h
;-----------------------------------------------------
```

Note: Set the bit 15 of register offset 0x65F10h of the Display Audio offset. Wait for the Codec to generate the wake event to the controller.

Follwing verbs should be send to the codec using the PIO method described in the below sections 9.1.3.

VerbTable0:

```
    ;Enable the third converter and Pin first (NID 08h)
     dd 20878101h
       //
// Audio Verb Table - 0x80862805
//
// Pin Widget 5 - PORT B
0x20571C10,
0x20571D00,
0x20571E56,
0x20571F18,
// Pin Widget 6 - PORT C
0x20671C20,
0x20671D00,
0x20671E56,
0x20671F18,
// Pin Widget 7 - PORT D
0x20771C30,
0x20771D00,
```

0x20771E56,

0x20771F18,

// Pin Widget 8 - PORT E -

0x20871C40,

0x20871D00,

0x20871E56,

0x20871F18

;disable the third converter and third Pin (NID 08h - upto gen9)

dd 20878100h

Codec Initialization Programming Sequence

After System BIOS has determined the presence of High Definition Audio codecs, it must follow the programming sequence below to update the codec with the correct jack information specific to the platform for the High Definition Audio driver to retrieve and use later.

There are two ways to send verbs to and receive response data from codecs over the High Definition Audio codec link: using CORB/RIRB (Command Output Ring Buffer / Response Input Ring Buffer) or using the Immediate Command/Immediate Response register pair. The sequence below uses the latter which does not require the availability of a memory buffer.

System BIOS should ensure that the High Definition Audio HDBAR D27:F0:10-17h contains a valid address value and is enabled by setting D27:F0:04h[1]. System BIOS must ensure program as mentioned in section 9.6, and then the Controller Reset# bit of Global Control register in memory-mapped space (HDBAR+08h[0]) is set to 1b and read back as 1b. Additional delay might be required to allow codec coming out of reset prior to subsequent operations, please contact your codec vendor for detail. When clearing this bit and setting it afterward, BIOS must ensure that minimum link timing requirements (minimum RESET# assertion time, etc.) are met.

Note: To initialize codec Bios should set the bit 15 of the register 0X65F10h of the Display MMIO to 1. This bit needs to be after the controller is brought out of reset. BIOS should wait for Controller to detect the wake event and recognize the Codec.

For each High Definition Audio codec present as indicated by HDBAR + 0Eh[3:0], System BIOS should perform the codec initialization as described below:

1. Read the VendorID/DeviceID pair from the attached codec.

- Verify that the ICB bit, HDBAR + 68h[0], is 0.
- Write verb 200F0000h (dword) to the IC register, HDBAR + 60h, where: ‘2’ (bits 31:28) represents the codec address (CAd).
- Program HDBAR + 68h[1:0] to 11b to send the verb to the codec.

- Poll the ICB bit, HDBAR+68h[0] until it returns 0 indicating the verb has been sent to the codec. BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- If HDBAR + 68h[1] = 1b indicating the response data from the codec is now valid, read HDBAR + 64h; the data is the VID/DID value returned by the codec.

2. Check against internal list to determine if there is a stored verb table which matches the CAd/VID/DID information.

Steps 1 and 2 are System BIOS implementation-specific steps and can be done in different ways. If a System BIOS has prior knowledge of a fixed platform/codec combination (e.g., for a System BIOS having 3 stored verb tables for 3 known codecs at known codec addresses on a known platform), a simple pre-defined codec-to-table matching can be used and steps 1 and 2 can be eliminated. For a System BIOS to support multiple codec/platform combinations, an internal match-list might be needed to match a platform/codec combination to a codec verb table.

3. If there is a match, send the entire list of verbs in the matching verb table one by one to the codec.

- Verify the ICB bit, HDBAR + 68h[0] is 0.
- Write the next verb (dword) in the table to HDBAR + 60h.
- Program HDBAR + 68h[1:0] to 11b to send the verb to codec.
- Poll the ICB bit, HDBAR + 68h[0] until it returns 0 indicating the verb has been sent to the codec. BIOS may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- Repeat the steps until all the verbs in the table have been sent.

Some verbs in the table may be dependent on certain platform-specific conditions. For example, for the sample table above, the verbs for Pin Complex 7 and 8 (NID=14,16 respectively) should be sent only if the Front Panel Jacks are present and connected on the platform, which may be indicated by a software flag that is controlled by a certain GPIO pin.

Audio Programming Sequence for Link wakeup

The following audio programming sequences are to be used for preventing the Unsolicited responses when 3 pin link is awake.

Display Audio codec generates a wake event whenever the power well (PGx - power well in which Display Audio codec HW resides) is powered up. If the link is already running, this wake event is considered as unsolicited response by audio controller in PCH. This may sometimes be considered as unnecessary URs. To avoid such URs following programming should be followed by SW. This sequence assumes communication between Audio and GFX drivers without HW to indicate Audio codec power well status.

Power down sequence:

1. Unplug event
2. PD goes low

3. PG2 low

4. Audio link should go low (there should be a communication between GFX driver and Audio Driver to turn off the link)

    1. To turn off iDisp-A link:
    2. Power off iDisp-A codec. Follow the PW2 turn off sequence.
    3. In HD Audio Controller (PCH) Program LCTL1.SPA = 0.
    4. Wait for LCTL1.CPA = 0 to indicate the link has clock stopped.

Power up sequence:

1. Plug event

2. PG2 goes high

3. Audio link to be enabled (there should be a communication between GFX driver and Audio Driver to turn on the link)

    1. To turn on iDisp-A link:
    2. In HD Audio Controller (PCH) Program LCTL1.SPA = 1.
    3. Wait for LCTL1.CPA = 1 to indicate the link has clock running.

4. Check WAKESTS[2] = 1 to indicate the codec wake up occurs

5. PD bit set

6. Codec awake. Continue with codec init.

Audio Programming Sequence

The following HDMI and DisplayPort audio programming sequences are to be used when enabling or disabling audio or temporarily disabling audio during a display mode set.

The audio codec and audio controller disable sequences must be followed prior to disabling the transcoder or port in a display mode set.

The audio codec and controller enable sequences can be followed after the transcoder is enabled and the port is enabled and completed link training (not sending training or idle patterns if DisplayPort).

The audio controller and audio codec sequences may be done in parallel or serial. In general, the change in ELDV/PD in the codec sequence will generate an unsolicited response to the audio controller driver to indicate that the controller sequence should start, but other mechanisms may be used. SW should make sure to set the Inactive (IA) bit to 0 before setting PD to 1.

Audio codec disable sequence:

- Disable sample fabrication

    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to "0".
- Disable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "1"
    - Set Upper_N_value and Lower_N_value (bits 27:20, 15:4) to all "0"s.
- Disable ELDV and ELD buffer
    - Set AUD_PIN_ELD_CP_VLD ELD_valid (bit 0, 4, or 8 based on which port is used) to "0"
- Wait for 2 vertical blanks
- Optional: Disable audio PD (Presence Detect)
    - Software may choose to skip this in order to keep PD enabled during a resolution switch.
    - Set AUD_PIN_ELD_CP_VLD Audio_Inactive (bit 3, 7, or 11) to "1". SW does not need to set this bit to enable Inactive bit.
    - Set AUD_PIN_ELD_CP_VLD Audio_Output_Enable (bit 2, 6, or 10) to "0".

Audio controller disable sequence:

- Program Stream ID to 0 - Verb ID 706
- Disable audio info frames transmission - Verb ID 732
- Disable Digen - Verb ID 70D
- Program the codec to D3 state if needed.
- Audio driver may stop the audio controller DMA engine at this point if needed, but not required.

Audio codec enable sequence:

- Enable audio Presence Detect
    - Set AUD_PIN_ELD_CP_VLD Audio_Inactive (bit 3, 7, or 11) to "0".
    - Set AUD_PIN_ELD_CP_VLD Audio_Output_Enable (bit 2, 6, or 10) to "1".
- Wait for 1 vertical blank
- Load ELD buffer and Enable ELDV
    - Set AUD_PIN_ELD_CP_VLD ELD_valid (bit 0, 4, or 8 based on which port is used) to "1".
- Enable timestamps
    - Set AUD_CONFIG N_value_index (bit 29) to "0" for HDMI or "1" for DisplayPort.
    - Set N_programming_enable (bit 28) to "0".
    - Program Upper_N_value and Lower_N_value (bits 27:20, 15:4) if a non-default N value is needed.
- Enable sample fabrication if this feature is needed
    - Set AUD_MISC_CTRL Sample_Fabrication_EN (bit 2) to "1".

Audio controller enable sequence:

- Program the codec to D0 state if in D3 state.

- Program Stream ID to non zero - Verb ID 706
- Enable audio info frames transmission - Verb ID 732
- Enable Digen - Verb ID 70D
- If audio controller DMA engine is stopped, audio driver can start the DMA engine at this point.

Audio Configuration

AUD_CONFIG

AUD_MISC_CTRL

AUD_VID_DID

AUD_RID

AUD_M_CTS_ENABLE

Audio Power State Format

AUD_PWRST

AUD_EDID_DATA

AUD_FREQ_CNTRL

AUD_INFOFR

AUD_PIN_PIPE_CONN_ENTRY_LNGTH

AUD_PIPE_CONN_SEL_CTRL

AUD_DIP_ELD_CTRL_ST

AUD_PIN_ELD_CP_VLD

Digital Display Interface

DDI Buffer

There is one instance of these registers per each DDI.

DDI_BUF_CTL

DDI_BUF_TRANS

DISPIO_CR_TX_BMU_CR0

DDI_BUF_TRANS and DISPIO_CR_TX_BMU_CR0 are programmed with the voltage swing values for each port. They can be configured in advance of the display mode set. The recommended values are listed below.

For DisplayPort, DDI_BUF_CTL is programmed during the mode set to select between the voltage swings pre-programmed in DDI_BUF_TRANS.

HDMI always uses the voltage swing programmed in DDI_BUF_TRANS entry 9.

I_boost

I_boost increases the I/O current, boosting the swing level. There are three I_boost values, in order of increasing boost; 0x1, 0x3, and 0x7.

I_boost can be configured separately for each port and can be enabled or disabled separately for each voltage swing level.

I_boost is configured by programming DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=<I_boost value; 0x1, 0x3, or 0x7>. It is then enabled when DDI_BUF_CTL selects a voltage swing entry that has DDI_BUF_TRANS_<entry> Balance Leg Enable=0x1. For DDIA with x4 capability (DDI_BUF_CTL DDIA Lane Capability Control = DDIA x4), the I_boost value has to be programmed in both tx_blnclegsctl_0 and tx_blnclegsctl_4.

The recommended buffer translation programming only uses I_boost for a few entries. A customer may request increased I_boost beyond the recommended values by specifying an I_boost value of 0x1, 0x3, or 0x7 to be applied to all swing entries for a port, then software needs to configure I_boost to that specified value and set DDI_BUF_TRANS_<all entries> Balance Leg Enable=0x1 for that port. This selection overrides the recommended I_boost settings.

Programming Note	
Context:	DDI Buffer
KBL re-uses SKL values, except where specific KBL values are listed.	

Recommended Buffer Translation Programming

KBL - Recommended buffer translation programming for DisplayPort											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level**[1]	**Pre-emphasis Level**[1]	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**KBL H and S**		**KBL Y**		**KBL U**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	400	400	0	0000 00A0h	0000 2016h	0000 00A1h	0000 1017h	0000 00A1h	0000 201Bh
1	0	1	400	600	3.5	0000 009Bh	0000 5012h	0000 0088h	0000 5012h	0000 0088h	0000 5012h
2	0	2	400	800	6	0000 0088h	0000 7011h	0000 00**CD** h	**8**000 7011h [3]	0000 00**CD** h	**8**000 7011h [3]
3	0	3	400	1000	9.5	0000 00C0h	**8**000 9010h [2]	0000 00C0h	**8**000 800Fh [3]	0000 00C0h	**8**000 9010h [3]
4	1	0	600	600	0	0000 009Bh	0000 2016h	0000 009Dh	0000 1017h	0000 009Dh	0000 201Bh

KBL - Recommended buffer translation programming for DisplayPort											
5	1	1	600	900	3.5	0000 0088h	0000 5012h	0000 00C0h	**8**000 5012h [3]	0000 00C0h	**8**000 5012h [3]
6	1	2	600	1000	6	0000 00C0h	**8**000 7011h [2]	0000 00C0h	**8**000 7011h [3]	0000 00C0h	**8**000 7011h [3]
7	2	0	800	800	0	0000 0097h	0000 2016h	0000 004Ch	0000 1017h	0000 004Fh	0000 2016h
8	2	1	800	1000	3.5	0000 00C0h	**8**000 5012h [2]	0000 00C0h	**8**000 5012h [3]	0000 00C0h	**8**000 5012h [3]

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

Recommended buffer translation programming for embedded DisplayPorts that support low voltage swings											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level[1]**	**Pre-emphasis Level[1]**	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**SKL/KBL H and S**		**SKL/KBL Y**		**SKL/KBL U**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	200	200	0	0000 00A8h	0000 0018h	0000 00A8h	0000 0018h	0000 00A8h	0000 0018h
1	0	1	200	250	1.5	0000 00A9h	0000 4013h	0000 00ABh	0000 4013h	0000 00A9h	0000 4013h
2	0	2	200	350	6	0000 00A2h	0000 7011h	0000 00A4h	0000 7011h	0000 00A2h	0000 7011h
3	0	3	200	600	9.5	0000 009Ch	0000 9010h	0000 00DFh	0000 9010h	0000 009Ch	0000 9010h
4	1	0	250	250	0	0000 00A9h	0000 0018h	0000 00AAh	0000 0018h	0000 00A9h	0000 0018h
5	1	1	250	350	3.5	0000 00A2h	0000 6013h	0000 00A4h	0000 6013h	0000 00A2h	0000 6013h
6	1	2	250	500	6	0000 00A6h	0000 7011h	0000 009Dh	0000 7011h	0000 00A6h	0000 7011h
7	2	0	350	350	0	0000 00ABh	0000 0018h	0000 00A0h	0000 0018h	0000 00ABh	0000 2016h
8	2	1	350	500	4.5	0000 009Fh	0000 7013h	0000 00DFh	0000 6012h	0000 009Fh	0000 5013h

Recommended buffer translation programming for embedded DisplayPorts that support low voltage swings											
9	Fail Safe	Fail Safe	800	800	0	0000 00DFh	0000 0018h	0000 008Ah	0000 0018h	0000 00DFh	0000 0018h

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

Recommended buffer translation programming for HDMI and DVI						
Non-Transition mV diff p-p	**Transition mV diff p-p**	**Pre-emphasis dB**	**SKL/KBL U, H, and S**		**SKL/KBL Y**	
			Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
400	400	0	0000 00ACh	0000 0018h	0000 00A1h	0000 0018h
400	600	3.5	0000 009Dh	0000 5012h	0000 00DFh	0000 5012h
400	800	6	0000 0088h	0000 7011h	0000 00CBh	8000 7011h[3]
450	450	0	0000 00A1h	0000 0018h	0000 00A4h	0000 0018h
600	600	0	0000 0098h	0000 0018h	0000 009Dh	0000 0018h
600	800	2.5	0000 0088h	0000 4013h	0000 0080h	0000 4013h
600	1000	4.5	0000 00CDh	8000 6012h[2]	0000 00C0h	8000 6013h[3]
800	800	0	0000 00DFh	0000 0018h	0000 008Ah	0000 0018h
[1]800	1000	2	0000 00CDh	8000 3015h[2]	0000 00C0h	8000 3015h[3]
1000	1250	2	0000 00C0h	8000 3015h[2]	0000 00C0h	**8**000 3015h[3]
1200	1200	0	0000 00C0h	8000 0018h[2]	0000 00C0h	8000 0018h[3]
Select one of the above values to program in DDI_BUF_TRANS entry 9.						

[1]This row is the recommended default if a specific value is not selected.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

The following values are customized for unique motherboard configurations selected through BIOS VBT.

Customized buffer translation programming for DisplayPort											
DDI_BUF_TRANS Entry Number	**Voltage Swing Level[1]**	**Pre-emphasis Level[1]**	**Non-Transition mV diff p-p**	**Transition mV diff p-p**	**Pre-emphasis dB**	**Custom 1 (Short ULT Traces)**		**Custom 2 (Long ULX Traces)**		**Custom 3 (KBL S Short Traces)**	
						Dword 1 [31:0]	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**	**Dword 1 [31:0]**	**Dword 0 [31:0]**
0	0	0	400	400	0	0000 00A2h	0000 1017h	0000 00A0h	**8**000 2016h[3]	0000 00A0h	0000 1017h
1	0	1	400	600	3.5	0000 0088h	0000 4013h	0000 0088h	**8**000 5012h[3]	0000 009Bh	0000 4013h
2	0	2	400	800	6	0000 00CDh	**8**000 6011h[2]	0000 00CDh	**8**000 7011h[3]	0000 0088h	0000 6011h

Customized buffer translation programming for DisplayPort											
3	0	3	400	1000	9.5	0000 00C0h	**8**000 800Fh[2]	0000 00C0h	**8**000 9010h[3]	0000 00C0h	**8**000 800Fh[2]
4	1	0	600	600	0	0000 009Dh	0000 1017h	0000 009Dh	**8**000 2016h[3]	0000 009Bh	0000 1017h
5	1	1	600	900	3.5	0000 00C0h	**8**000 4013h[2]	0000 00C0h	**8**000 5012h[3]	0000 0088h	0000 4013h
6	1	2	600	1000	6	0000 00C0h	**8**000 6011h[2]	0000 00C0h	**8**000 7011h[3]	0000 00C0h	**8**000 6011h[2]
7	2	0	800	800	0	0000 0088h	0000 1017h	0000 0088h	**8**000 2016h[3]	0000 0097h	0000 1017h
8	2	1	800	1000	3.5	0000 00C0h	**8**000 4013h[2]	0000 00C0h	**8**000 5012h[3]	0000 00C0h	**8**000 4013h[2]

[1]The voltage swing level and pre-emphasis level values follow the naming used in the DisplayPort standard.

[2]I_boost needs to be configured to recommended level 0x1. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x1.

[3]I_boost needs to be configured to recommended level 0x3. DISPIO_CR_TX_BMU_CR0 tx_blnclegdisbl=0x00 and tx_blnclegsctl_<selected DDI>=0x3.

DDI AUX Channel

DDI_AUX_CTL

DDI_AUX_DATA

DDI_AUX_MUTEX

AUX programming sequence

A general purpose AUX functional programming sequence is provided below.

AUX Functional Sequence

Step	Description	Register	Notes
1	Display must already be initialized.		Power well1 enabled cdclk enabled IO powered up as needed for Aux on this project
1b	Power well containing Aux logic powered up	PWR_WELL_CTL	High level power well partitioning shown in display overview diagram
2	Disable DC5 and DC6 before a DDI A AUX channel transaction is sent.	DC_STATE_EN	PG1 may disable automatically for DC5 or DC6.
Skip step 3, 4 if PSR1/SRD, PSR2 or GTC are NOT enabled. Do not skip step 3,4 if PSR1/SRD, PSR2 or GTC are enabled.			
3	Enable MUTEX without changing MUTEX status	DDI_AUX_MUTEX[31]='1'	
4	Read MUTEX status	DDI_AUX_MUTEX[30]	If MUTEX status is '1', wait for 500 us and poll for MUTEX status == '0'.
5	Program AUX data registers.	DDI_AUX_DATA_*_[0-4]	
6	Program control to configure AUX and START transaction.	DDI_AUX_CTL_*	Timeout timer value must be at least 600us. Timer values: 0: 400us 1: 600 us 2: 800 us 3: 1600 us START trigger: DDI_AUX_CTL_*[31]='1'
7	Wait for AUX transaction complete.		AUX Transaction complete interrupt if set OR when DDI_AUX_CTL_*[31:30] = '01'.
8	Check that receive data has no errors	DDI_AUX_CTL_*[25]	If set: write a '1' to clear this bit and skip reading AUX data registers.
9	Read AUX data register	DDI_AUX_DATA_*_[0-4]	Condition: Aux Channel Control Register Send/Busy bit is NOT asserted
10	Clear status flags	DDI_AUX_CTL_*[30]	Transaction done status

Step	Description	Register	Notes
Skip step 11 if PSR1/SRD, PSR2 or GTC are NOT enabled. Do not skip step 11 if PSR1/SRD, PSR2 or GTC are enabled.			
11	Release MUTEX	DDI_AUX_MUTEX[31:30]= "11" OR DDI_AUX_MUTEX[31:30]="01"	Release with (is optional) or without disabling MUTEX function.

DisplayPort Transport

There is one instance of these registers per each DDI.

DP_TP_CTL

DP_TP_STATUS

Global Time Code (GTC)

Top Level GTC

GTC_CTL

GTC_DDA_M

GTC_DDA_N

GTC_LIVE

GTC Interrupt Bit Definition

GTC_IMR

GTC_IIR

DDI Level GTC

GTC_PORT_CTL
GTC_PORT_TX_CURR
GTC_PORT_TX_PREV
GTC_PORT_MISC

GTC Target Frequency Selection

For GTC top level logic, CDCLK is taken as an input and scaled to a "target frequency" which has a period which is an exact multiple of 0.5ns. This period is also known as the accumulator increment.

Once a target frequency + accumulator increment is selected, an M and N value can be picked and fine tuned to achieve the scaling from CDCLK to target frequency.

In order for the accumulated GTC Live value to match exactly with the real passage of time, the following must be true:

The target frequency selected must be the CLOSEST possible selection to CDCLK. This corresponds to rounding the accumulator increment to the NEAREST 0.5ns increment with respect to the period of CDCLK.

Example

	Frequency	Period
Given CDCLK	337.500 MHz	2.96 ns
1st closest target frequency/increment	333.333 MHz	3.00 ns
2nd closest target frequency/increment	400.000 MHz	2.50 ns
3rd closest target frequency/increment	285.714 MHz	3.50 ns
4th closest target frequency/increment	500.000 MHz	2.00 ns

Only using the 1st closest target frequency 333.333MHz / accumulator increment of 3.00 ns will result in the GTC Live Value correctly tracking the real passage of time.

Example Calculation Flow:

1. Find the period of CDCLK -- for 337.5 MHz = 2.96 ns.
2. Round to the nearest 0.5 ns -- 3.00 ns (this is your accumulator increment value).
3. Find the “target frequency” from the rounded period value of step 2 -- 333.333 MHz is the “target frequency”.
4. Find the ratio of target frequency / CDCLK = 0.987654321.
5. Choose M and N to satisfy M / N = same ratio as step 4.

South Display Engine Registers

The South Display Engine supports Hot Plug Detection, GPIO, GMBUS, Panel Power Sequencing, and Backlight Modulation.

Terminology

Access Field	Description	Should be implemented as
R/W (Read/Write)	The value written into this register will control hardware and is the same value that will be read.	Write data is stored. Read is from the stored data. Stored value is used to control hardware.
Reserved	Unused register bit. Don't assume a value for these bits. Writes have no effect.	Write data is ignored. Read is zero.
MBZ (Must Be Zero)	Always write a zero to this register.	May be implemented as Reserved or as R/W.
PBC (Preserve Bit Contents)	Software must write the original value back to this bit. This allows new features to be added using these bits.	May be implemented as Reserved or as R/W.
Read Only	The read value is determined by hardware. Writes to this bit have no effect.	Write data is ignored. Read is from a status signal or some other internal source.
Write Only	The value written into this register will control hardware. Reads return zero.	Write data is stored. Read is zero. Stored value is used to control hardware.
R/W Clear (Read/Write Clear)	Sticky status bit. Hardware will set the bit, software can clear it with a write of 1b.	Internal hardware events set a sticky bit. Read is from the sticky bit. A write of 1b clears the sticky bit.
Double Buffered	Write when desired and the written value will take effect at the time of the double buffer update point. Reads will return the written value, which is not necessarily the value being currently used to control hardware. Some have a specific arming sequence where a write to another register is required before the update can take place. This is used to ensure atomic updates of several registers.	Two stages of registers used. Write data is stored into first stage. Read is from the first stage stored data. First stage stored value is transferred to second stage storage at the double buffer update point. Second stage stored value is used to control hardware. Arm/disarm logic may be used for some registers to control the double buffer update point.
Write/Read Status	The value written into this register will control hardware. The read value is determined by hardware.	Write data is stored. Stored value is used to control hardware. Read is from a status signal or some

Access Field	Description	Should be implemented as
		other internal source.

Shared Functions

Fuses and Straps

SFUSE_STRAP

Raw Clock

RAWCLK_FREQ must be programmed to match the raw clock frequency.

Description
RAWCLK_FREQ

Description
Raw clock frequency = 24 MHz

Interrupts and Hot Plug

Description
South Display Engine Interrupt Bit Definition **SINTERRUPT** **SHPD_PULSE_CNT** **SHPD_FILTER_CNT**
SHOTPLUG_CTL **SHOTPLUG_CTL2**

Panel Power and Backlight

Panel Power

Description
PP_DIVISOR
PP_STATUS **PP_CONTROL** **PP_ON_DELAYS** **PP_OFF_DELAYS**

Backlight

This section refers to the PCH display backlight control. For CPU display backlight control, see North Display Engine Registers.

The backlight PWM output frequency is determined by the PWM clock frequency, increment, and frequency divider.

PWM output frequency = PWM clock frequency / PWM increment / PWM frequency divider

The frequency divider minimum must be greater than or equal to the number of brightness levels required by software; typically 100 or 256.

Description
PWM clock frequency = 24 MHz PWM increment = 128 or 16, selectable by software PWM frequency divider maximum = 65,536 PWM output frequency range with 100 brightness levels and increment 128 = 3 to 1,875 Hz PWM output frequency range with 100 brightness levels and increment 16 = 23 to 15,000 Hz PWM output frequency range with 256 brightness levels and increment 128 = 3 to 732 Hz PWM output frequency range with 256 brightness levels and increment 16 = 23 to 5,859 Hz
PWM clock frequency = Raw clock frequency = 19.2, 24, or 38.4 MHz. Value is found in SFUSE_STRAP. PWM increment = 1 PWM frequency divider maximum = 2^32 PWM output frequency range with 100 brightness levels and 19.2 MHz clock = 0.0004 to 192,000 Hz

Description
PWM output frequency range with 100 brightness levels and 24 MHz clock = 0.0006 to 240,000 Hz
PWM output frequency range with 100 brightness levels and 38.4 MHz clock = 0.0009 to 384,000 Hz
PWM output frequency range with 256 brightness levels and 19.2 MHz clock = 0.0004 to 75,000 Hz
PWM output frequency range with 256 brightness levels and 24 MHz clock = 0.0006 to 93,750 Hz
PWM output frequency range with 256 brightness levels and 38.4 MHz clock = 0.0009 to 150,000 Hz

Backlight Enabling Sequence

Description
1. Set frequency and duty cycle in SBLC_PWM_CTL2 Backlight Modulation Frequency and Backlight Duty Cycle.
2. Set granularity in 0xC2000 bit 0 (0 = 16, 1 = 128).
3. Enable PWM output and set polarity in SBLC_PWM_CTL1 PWM PCH Enable and Backlight Polarity.
...
4. Change duty cycle as needed in SBLC_PWM_CTL2 Backlight Duty Cycle.

If needed, granularity, polarity, and override can be programmed earlier than shown.

Backlight Registers

Description
SBLC_PWM_CTL1
SBLC_PWM_CTL2

GMBUS and GPIO

Registers

Description
GPIO_CTL - GPIO Control
GMBUS0 - GMBUS Clock/Port Select
GMBUS1 - GMBUS Command/Status
GMBUS2 - GMBUS Status
GMBUS3 - GMBUS Data Buffer
GMBUS4 - GMBUS Interrupt Mask
GMBUS5 - GMBUS 2 Byte Index

Pin Usage

These GPIO pins allow the support of simple query and control functions such as DDC interface protocols. The GMBUS controller can be used to run the interface protocol, or the GPIO pins can be manually programmed for a "bit banging" interface.

The following tables describe the expected GPIO pin to register mapping. OEMs have the ability to remap these functions onto other pins as long as the hardware limitations are observed. The GPIO pins may also be muxed with other functions such that they are only available when the other function is not being used.

Port #	Name	Pin	Pull up/down	Description
5	DDID CTLDATA	DDPD_CTRLDATA	No (Weak pull down on reset)	DDC for port D. Digital port D present strap is set if pin is 1 at rising edge of PCH_PWROK.
	DDID CTLCLK	DDPD_CTRLCLK	No	
4	DDIB CTLDATA	DDPB_CTRLDATA	No (Weak pull down on reset)	DDC for port B. Digital port B present strap is set if pin is 1 at rising edge of PCH_PWROK.
	DDIB CTLCLK	DDPB_CTRLCLK	No	
3	DDIC CTLDATA	DDPC_CTRLDATA	No (Weak pull down	DDC for port C. Digital port C present strap is set if pin is 1 at rising edge of PCH_PWROK.

			on reset)	
	DDIC CTLCLK	DDPC_CTRLCLK	No	

GPIO Programming for I2C Bit Bashing

To drive GPIO pin low, program direction to "out" and data value to "0", along with mask bits.

To drive GPIO pin high (tristate to allow external pull up to activate), program direction to "in", along with mask bit. No need to set data value to "1".

GMBUS Controller Programming Interface

The GMBUS (Graphic Management Bus) is used to access/control devices connected to the GPIO pins.

Basic features:

1. I^2C compatible.
2. Bus clock frequency of 50 KHz or 100 KHz.
3. Attaches to any of the GPIO pin pairs.
4. 7-bit or 10-bit Slave Address and 8-bit or 16-bit index.
5. Double buffered data register and a 9 bit counter support 0 byte to 256 byte transfers.
6. Supports stalls generated by the slave device pulling down the clock line (Slave Stall), or delaying the slave acknowledge response.
7. Status register indicates error conditions, data buffer busy, time out, and data complete acknowledgement.
8. Detects and reports time out conditions for a stall from a slave device, or a delayed or missing slave acknowledge.
9. Interrupts may optionally be generated.
10. Does not directly support segment pointer addressing as defined by the Enhanced Display Data Channel standard.

Segment pointer addressing as defined by the Enhanced Display Data Channel standard:

1. Use bit bashing (manual GPIO programming) to complete segment pointer write **without terminating in a stop or wait cycle**.
2. Terminate bit bashing phase with both I^2C lines pulled high by tri-stating the data line before the clock line. Follow EDDC requirement for response received from slave device.
3. Initiate GMBUS cycle as required to transfer EDID following normal procedure.

Sequence for GMBUS Burst Reads Greater Than 511 Bytes

- Not available until KBLPCH-H A0, SPT-LP D1, SPT-H E1

1. X=number of bytes to read.
2. Set byte count override bit GMBUS0 bit 6.
3. Set total byte count field in GMBUS1 to 511.
4. Read X bytes.
5. Issue GMBUS stop by writing 100b to GMBUS1 Bus Cycle Select.
6. Clear byte count override bit.

Example with 526 bytes

1. X=526.
2. Set byte count override bit GMBUS0 bit 6.
3. Set total byte count field in GMBUS1 to 511.
4. Read 526 bytes.
5. Write 100b to GMBUS1 Bus Cycle Select.

   Clear byte count override bit.

Display Watermark Programming

Watermark Overview

The display watermarks are used to control the display engine memory request behavior.

Description
The default settings of the watermark configuration registers will **not** allow the display engine to operate. The watermark values must be properly calculated and programmed in order to enable a display and achieve optimum power and performance. Incorrectly programmed watermark values can result in screen corruption.

The watermarks should be calculated and programmed when any of the watermark calculation inputs change. This includes planes enabling or disabling, plane source format or size changing, etc.

Besides programming the watermark registers, there are other display configuration requirements and registers that must be programmed in order for the display to operate in a low power mode, and there are memory controller configuration requirements which are not documented here.

Watermark Calculations

The display watermarks are calculated using information from the display configuration and memory latencies. The watermarks must be calculated and programmed before enabling a plane or changing a plane configuration.

For YUV 420 Planar formats, only the Y surface watermark value is calculated and programmed. Though the UV surface watermark value is not programmed seperately, it must be calculated to make sure that the UV buffer allocation satisfies the latency requirments.

The ceiling function rounds any non-integer value up to the next greater integer. Example: ceiling[0.3]=1, ceiling[2.1]=3, ceiling[4.8]=5, ceiling[4]=4

Watermark Algorithm

1. Retrieve memory latency values
    - See the Memory Values section to find the memory latency values
    - The memory values do not change after boot, so software may cache them to avoid re-reading
2. For each enabled pipe (run each time pipe configuration changes)
    A. Calculate adjusted pipe pixel rate
        I. Adjusted pipe pixel rate = pixel rate for the screen resolution
            - If there will be dynamic switching between refresh rates, either use the fastest pixel rate, or re-calculate using the current pixel rate when the refresh rate is switched
            - If plane 90 or 270 rotation is enabled, use the rotated width and height in pixel rate calculations.
        II. If TRANS_CONF Interlaced Mode == PF-ID, adjusted pipe pixel rate = adjusted pipe pixel rate * 2
        III. If pipe scaling enabled, adjusted pipe pixel rate = adjusted pipe pixel rate * pipe down scale amount
            - See the Scaling section to find the down scale amount
    B. Program WM_LINETIME Line Time = roundup[8 * pipe horizontal total pixels / adjusted pipe pixel rate MHz]
3. For each enabled plane (run each time pipe or plane configuration changes)
    A. Calculate adjusted plane pixel rate
        I. Adjusted plane pixel rate = adjusted pipe pixel rate
        II. If plane scaling enabled, adjusted plane pixel rate = adjusted plane pixel rate * plane down scale amount
            - See the Scaling section to find the down scale amount
    B. For each valid memory latency level

Plane Bytes per pixel	Minimum Scanlines for Y Tile	
	0/180 Rotation	90/270 Rotation
1 or NV12	4	16
2, P0xx, or YUV422	4	8
4	4	4
8	4	N/A

        I. Calculate method 1
            - Method 1 = memory latency microseconds * adjusted plane pixel rate MHz * plane source bytes per pixel / 512
        II. Calculate method 2

- plane bytes per line = plane source width pixels * plane source bytes per pixel
- Calculate plane blocks per line
  - If plane memory format is Linear
    - plane blocks per line = ceiling[plane bytes per line / 512] + 1
  - Else If plane memory format is Y tile
    - plane blocks per line = ceiling[Minimum Scanlines for Y tile * plane bytes per line / 512]/Minimum Scanlines for Y tile
  - Else
    - plane blocks per line = ceiling[plane bytes per line / 512]
- Method 2 = ceiling[(memory latency microseconds * adjusted plane pixel rate MHz) / Pipe horizontal total number of pixels] * plane blocks per line

III. Calculate Y tile minimum

- Y tile minimum = Minimum Scanlines for Y tile * plane blocks per line

IV. Select the watermark result

- If plane memory format is X tile or linear
  - If (((plane source bytes per pixel * pipe horizontal total number of pixels) / 512) < 1) AND ((plane bytes per line / 512) < 1) // Special case for unrealistically small horizontal total
    - Selected Result Blocks = Method 2
  - Else If (plane buffer allocation / plane blocks per line) >=1
    - Selected Result Blocks = minimum[Method 1, Method 2]
  - Else
    - Selected Result Blocks = Method 1
- Else // Y tile
  - Selected Result Blocks = maximum[Method 2, Y tile minimum]
-

V. Convert result to blocks and lines

- Result Blocks = ceiling[Selected Result Blocks] + 1
- Result Lines = ceiling[Selected Result Blocks / plane blocks per line]

this applies to all Gen9 projects

If latency level 1 through 7 and Y tile:

Result Blocks = Result Blocks + Y tile minimum

Result Lines = Result Lines + Minimum Scanlines for Y tile

If latency level 1 through 7 and not Y tile:

- Result Blocks = Result Blocks + 1

This applies to all Gen9 projects

If Render Decompression enabled and latency level 0:

Result Blocks = Result Blocks + Y tile minimum

// For latency levels 1 through 7 ensure that the their result lines/blocks >= latency level 0 lines/blocks.

For latency levels 1 through 7

If latency level result blocks < latency level 0 blocks

latency level result blocks = latency level 0 blocks

If latency level result lines < latency level 0 lines

- latency level result lines = latency level 0 lines

VI. Compare against the maximum
- If (Result Blocks >= plane buffer allocation), maximum exceeded for this latency level
- If latency level 1 through 7
  - If (Result Lines > 31), maximum exceeded for this latency level
- For YUV 420 Planar formats, perform the above check for both Y and UV planes.

4. For transition watermark

A. Calculate transition offset
- Transition Offset Blocks = Transition minimum + Transition amount
- See Transition Watermark section for transition minimum and transition amount

B. Calculate transition Y tile minimum
- Transition Y tile minimum = 2 * memory latency level 0 Y tiled minimum

C. Select the watermark result
- If plane memory format is X tile or linear
  - Result Blocks = Memory latency level 0 Selected Result Blocks + Transition Offset Blocks
- Else // Y tile
  - Result Blocks = maximum[Memory latency level 0 Selected Result Blocks, Transition Y tile minimum] + Transition Offset Blocks

D. Convert result to blocks
- Result Blocks = ceiling[Result Blocks] + 1

This applies to all Gen9 projects

If not Y tile:

- Result Blocks = Result Blocks + 1
-

E. Compare against the maximum

- If (Result Blocks >= plane buffer allocation), maximum exceeded for transition watermark
- For YUV 420 Planar formats, perform the above check for both Y and UV planes.

5. Program watermark registers
   A. For each latency level 0 to 7
      - If memory latency for this level is invalid, or the maximum was exceeded for this level or any previous level, program PLANE_WM_<latency level> Enable = 0
        - **If watermark latency level 0 exceeds the maximum, the plane must not be enabled.**
      - Else program PLANE_WM_<latency level> Enable = 1, Lines = Result Lines, Blocks = Result Blocks
        - The latency level 0 Lines value is ignored by hardware in Gen9
   B. For transition watermark
      - If the maximum was exceeded for the transition watermark, program PLANE_WM_TRANS Enable = 0
      - Else program PLANE_WM_TRANS Enable = 1, Blocks = Transition Result Blocks
        - The transition watermark Lines value is ignored by hardware
   C. Write the plane surface base address register to trigger update of the watermarks and other plane double buffered registers. This should be done only after all plane configuration is configured to match the new watermark values.

Transition Watermark

The transition watermark is used for Isochronous Priority Control (IPC). When IPC is enabled (ARB_CTL2 Enable IPC), plane read requests are sent at high priority until filling above the transition watermark, then the requests are sent at lower priority until dropping below the level 0 watermark. The lower priority requests allow other memory clients to have better memory access. If the transition watermark is not enabled, the plane behaves as if the transition watermark was programmed to the top of the plane buffer allocation. When IPC is disabled, all plane read requests are sent at high priority.

The transition watermark is programmed as a tunable amount above the level 0 watermark. Tuning to higher values will tend to cause longer periods of high priority reads followed by longer periods of lower priority reads. Tuning to lower values will tend to cause shorter periods of high and lower priority reads. The exact behavior depends on the memory bandwidth, display bandwidth, and other memory traffic in the system.

The transition watermark has a minimum value to ensure the demote does not happen before enough data has been read to meet the level 0 watermark requirements.

Transition Minimum
14 Blocks

Scaling

A scaler (pipe or plane scaler) is down scaling when it is enabled and the scaler input size is greater than the scaler output size.

Down scaling effectively increases the pixel rate. Up scaling does not reduce the pixel rate.

For plane scaling, the scaler input size is the plane size and the output size is the scaler window size.

For pipe scaling, the scaler input size is the pipe source size and the output size is the scaler window size.

Horizontal down scale amount = maximum[1, Horizontal source size / Horizontal destination size]

Vertical down scale amount = maximum[1, Vertical source size / Vertical destination size]

Total down scale amount = Horizontal down scale amount * Vertical down scale amount

Workaround	
Context:	Watermark Calculations

Workaround when calculating watermarks

Calculate arbitrated display bandwith:

For each pipe {

For each plane, except cursor, enabled on the pipe {

Plane bandwidth MB/s = pixel rate MHz * source pixel format in bytes * plane down scale amount * pipe down scale amount

}

Pipe bandwidth MB/s = Number of planes, except cursor, enabled on the pipe * Max(bandwidth from each plane) // find plane with highest bandwidth and multiply with number of enabled planes

}

Arbitrated display bandwidth = Number of pipes enabled * Max(bandwidth from each pipe) // find pipe with highest bandwidth and multiply with number of enabled pipes

If there is any Ytile plane enabled and arbitrated display bandwidth > 20% of raw system memory bandwidth (# memory channels * memory frequency * channel width in bytes) {

Double Ytile planes minimum lines and program all watermark levels accordingly. Do not change the latency values.

Increase Xtile planes watermark latency for all levels by 15us

}

If memory is dual channel single rank, Xtile limit = 35%, else Xtile limit = 60%

If there is no Ytile plane enabled and arbitrated display bandwidth > (Xtile limit * raw system memory bandwidth) {

Increase Xtile planes watermark latency for all levels by 15uS

}

System Agent Geyserville (SAGV)

On core CPUs, SAGV dynamically adjusts the system agent voltage and clock frequencies depending on power and performance requirements. The display engine access to system memory is blocked during the adjustment time.

SAGV defaults to enabled. Software must use the GT-driver pcode mailbox to disable SAGV when the display engine is not able to tolerate the blocking time.

See the Memory Values section to find the SAGV block time.

Requirement before plane enabling or configuration change: Disable SAGV if any enabled plane will not be able to enable watermarks for memory latency >= SAGV block time, or any transcoder is interlaced. Else, enable SAGV.

If software ensures single pipe configurations always have enough data buffer allocation to tolerate SAGV, it can then simply disable SAGV anytime multiple display pipes are enabled or interlace is enabled, and re-enable SAGV when switching back to a single, non-interlaced pipe.

Sequence to Disable SAGV

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low = 0x00000000.
3. Write GT Driver Mailbox Interface = 0x80000021.
4. Poll for GT Driver Mailbox Interface Run/Busy == 0x0.
    - Timeout and fail after 150 us.
5. Read GT Driver Mailbox Data Low, if bit 0 == 0x1, continue, else go to step 2.
    - If the condition in step 5 is not satisfied after cycling through steps 2-5 for 1 ms (typically <200 us), timeout and fail.
6. Continue with plane and pipe programming.

Sequence to Enable SAGV

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data Low = 0x00000003.
3. Write GT Driver Mailbox Interface = 0x80000021.
4. Continue with plane and pipe programming. There is no need to wait for SAGV enabling to complete.

Examples

Example pixel rate adjustments:

Pixel rate for screen resolution is 130 MHz. No interlacing. Pipe scale 1920x1080 pipe source size to 1714x1120 scaler window size. Plane scale 1920x1080 plane size to 800x600 scaler window size.

Pipe horizontal down scale amount = maximum[1, 1920 / 1714] = 1.12

Pipe vertical down scale amount = maximum[1, 1080 / 1120] = 1 **// Max condition was hit**

Pipe total down scale amount = 1.12 * 1 = 1.12

Adjusted pipe pixel rate = 130 MHz * 1.12 = 145.6 MHz

Plane horizontal down scale amount = maximum[1, 1920 / 800] = 2.4

Plane vertical down scale amount = maximum[1, 1080 / 600] = 1.8

Plane total down scale amount = 2.4 * 1.8 = 4.32

Adjusted plane pixel rate = 145.6 MHz * 4.32 = 628.99 MHz

Example method, block, and line calculations:

Plane source 4 Bpp, Plane X tile, Plane source width 1920 pixels, Horizontal total 2200 pixels, Adjusted plane pixel rate 148.5 MHz, memory latency 7.5 us

Method 1 = 148.5 MHz * 4 Bpp * 7.5 us / 512 = 8.7 blocks

Plane bytes per line = 1920 pixels * 4 Bpp = 7680 Bytes/line

Plane blocks per lines = ceiling[7680 / 512] = 15 blocks

Method 2 = ceiling[(7.5 us * 148.5 MHz) / 2200 pixels] * 15 blocks = 15 blocks

Y tile minimum = 4 * 15 blocks = 60 blocks

Result Blocks = minimum[8.7 blocks, 15 blocks] = 8.7 blocks // X tile so does not use Y tile minimum

Result Blocks = ceiling[8.7 blocks] + 1 block = 10 blocks

Result Lines = ceiling[8.7 blocks / 15] = 1 lines

Memory Values

Retrieve Memory Latency Data

1. Ensure any previous GT Driver Mailbox transaction is complete.
2. Write GT Driver Mailbox Data0=0x0000_0000 (first set of latency values) and GT Driver Mailbox Data1=0x0000_0000
3. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
4. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
5. Read GT Driver Mailbox Data0 for the first set of memory latency values
6. Write GT Driver Mailbox Data0=0x0000_0001 (second set of latency values) and GT Driver Mailbox Data1=0x0000_0000
7. Write GT Driver Mailbox Interface Run/Busy=1, Address Control=All 0s, Command/Error Code=06h
8. Poll GT Driver Mailbox Interface for Run/Busy indication=0b and Command/Error Code=00h (success)
    - Timeout after 100 us and do not enable display planes.
9. Read GT Driver Mailbox Data0 for the second set of memory latency values

Memory Latency Data Definition

First Set		
Data0 Bit	**Name**	**Description**
31:24	Level 3	Number of microseconds for level 3.
23:16	Level 2	Number of microseconds for level 2.
15:8	Level 1	Number of microseconds for level 1.
7:0	Level 0	Number of microseconds for level 0.

Second Set		
Data0 Bit	**Name**	**Description**
31:24	Level 7	Number of microseconds for level 7.
23:16	Level 6	Number of microseconds for level 6.
15:8	Level 5	Number of microseconds for level 5.
7:0	Level 4	Number of microseconds for level 4.

If level 1 or any higher level has a value of 0x00, that level and any higher levels are unused and invalid, so the associated watermark registers must not be enabled.

It is allowed to have the same value in adjacent levels.

Workaround	
Context:	Display Watermark Programming
The mailbox response data may not account for memory read latency. If the mailbox response data for level 0 is 0us, add 2 microseconds to the result for each valid level.	

SAGV Block Time

SAGV Block Time
30 us

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 13: Memory-mapped Input/Output (MMIO)

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Slice Registers and Die Recovery **1**

SW Virtualization Reserved MMIO range **1**

Slice Registers and Die Recovery

When slice 0 is disabled (for example, GT3 fused to GT2 with a slice 0 fault), any read to a slice-located MMIO register must be directed to slice 1, otherwise data of '0' will be returned. This applies to SRM cycles from any command streamer.

MMIO Range Start	MMIO Range End	Unit Description
00005500	00005FFF	WMBE
00007000	00007FFF	SVL
00009400	000097FF	CP unit reg. file - Copy in Slice Common (in all slices)
0000B000	0000B0FF	L3 unique status registers for each slice (unicast per GT).
0000B100	0000B3FF	L3 bank config space (multicast copy per bank and slice)
0000E000	0000E0FF	DM
0000E100	0000E1FF	SC
0000E200	0000E3FF	GWL (inst. 0)
0000E200	0000E3FF	GWL (inst. 1)
0000E200	0000E3FF	GWL (inst. 2)
0000E400	0000E7FF	TDL

SW Virtualization Reserved MMIO range

The MMIO address range from 0x178000 thru 0x178FFF is reserved for communication between a VMM and the GPU Driver executing on a Virtual Machine.

HW does not actually implement anything within this range. Instead, in a SW Virtualized environment, if a VM driver issues a read to this MMIO address range, the VMM will trap that access, and provide whatever data it wishes to pass to the VM driver. In a non-SW-Virtualizated environment (including an SR-IOV Virtualized environment), reads will return zeros, like any other unimplemented MMIO address. Writes to this range are always ignored.

It is important that no "real" HW MMIO register be defined within this range, as it would be inaccessable in a SW-virtualized environment.

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 14: Observability

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Observability Overview **1**

Trace **1**

Performance Visibility 1

Motivation For Hardware-Assisted Performance Visibility 1

Performance Event Counting 1

OA Programming 3

HW Support 3

Performance Counter Registers 3

OA Interrupt Control Registers 6

Performance Counter Report Formats 7

Performance Counter Reporting 12

Aggregating Counters 13

Flexible EU Event Counters 16

MI_REPORT_PERF_COUNT ..... **Error! Bookmark not defined.**

Observability Overview

As GFX-enabled systems and usage models have grown in complexity over time, a number of hardware features have been added to provide more insight into hardware behavior while running a commercially available operating system. This chapter documents these features with pointers to relevant sections in other chapters. Supported observability features include:

Feature
Performance counters
Internal node tracing

Note: This chapter describes the registers and instructions used to monitor GPU performance. Please review other volumes in this specification to understand the terms, functionality and details for specific Intel graphics devices.

Trace

This section contains the following contents:

Feature
• Performance Visibility

Performance Visibility

Motivation For Hardware-Assisted Performance Visibility

As the focus on GFX performance and programmability has increased over time, the need for hardware (HW) support to rapidly identify bottlenecks in HW and efficiently tune the work sent to same has become correspondingly important. This part of the BSpec describes the HW support for Performance Visibility.

Performance Event Counting

An earlier generation introduced dedicated GFX performance counters to address key issues associated with existing chipset CHAPs counters (lack of synchronization with GFX rendering work and low sampling frequency achievable when sampling via CPU MMIO read). Furthermore, reliance on SoC assets created a cross-IP dependency that was difficult to manage well. Hence, the approach since that earlier generation has been to use dedicated counters managed by the graphics device driver for graphics performance measurement. The dedicated counter values are written to memory whenever an MI_REPORT_PERF_COUNT command is placed in the ring buffer.

While this approach eliminated much of the error associated with the previous approaches, it is still limited to sampling the counters only at the boundaries between ring commands. This inherently limited the ability of performance analysis tools to drill down into a primitive, which can contain thousands of triangles and require several hundreds of milliseconds to render.

Additionally, Intel design and architecture teams found that the existing silicon-based performance analysis tools provided only a general idea of where a problem may exist but were not able to pin point a problem. This was generally because the counter values are integrated across a very large time period, washing out the dynamic behavior of the workload.

Gen7 enhanced the aggregating counters to support the additional thread types generated by DX11 workloads. The high rate at which interesting internal events can occur motivated adding an interrupt-generation capability so that HW could notify SW when the data buffer was approaching full.

Gen7.5 enhances support for high reporting frequencies by increasing the report buffer size in order to allow SW sufficient time between performance monitoring interrupts, enabling single run histogramming support for events like pixels per polygon. Desire for more flexibility in custom event creation drove addition of 4 more Boolean counters.

Issues with SNB support drove enhancements to enable performance monitoring with RC6 enabled, different report buffer ring wrap behavior, and MMIO visibility into performance counters.

Gen8 enhances functionality of aggregating counters for EUs by providing some flexibility in what quantities are aggregated across all EUs including more quantities relevant to GPGPU workloads. Since several of the previously defined aggregating counters had not delivered very much value on earlier projects, the overall number of A-counters has gone down even though aggregating counters for sampler/pixel-level functionality have been added/redefined. Custom counter creation has been enhanced by adding the ability to negate a signal at the input of the Boolean logic. Given that the increased complexity of GFX workloads and number of EUs in GT2/GT3 could lead to more frequent counter overflows, the width A-counters has increased to 40 bits. HW optimizations have also modified the SW interface slightly.

OA Programming Guidelines

In order for OA counters to increment the 'Counter Stop-Resume Mechanism' bit of the OACTXCONTROL register must be set. This requires a RCS context with this bit set be loaded, and either RCS force wake be enabled or the RCS context be left active for the duration of the window this counter is needed for.

If GT is running with decoupled unslice / slice frequencies, the following restrictions apply:

- C counters cannot be used for counting slice events.
- Max of 7 slice signals can be counted together.

Note that counting unslice events is unaffected.

HW Support

This section contains various reporting counters and registers for hardware support for Performance Visibility.

Performance Counter Registers

Register
OACONTROL - Observation Architecture Control
OACTXCONTROL - Observation Architecture Control per Context
OACTXID - Observation Architecture Control Context ID
OA_IMR - OA Interrupt Mask Register
OASTATUS - Observation Architecture Status Register
OAHEADPTR - Observation Architecture Head Pointer
OATAILPTR - Observation Architecture Tail Pointer
OABUFFER - Observation Architecture Buffer
OASTARTTRIG_COUNTER - Observation Architecture Start Trigger Counter
OARPTTRIG_COUNTER - Observation Architecture Report Trigger Counter
OAREPORTTRIG2 - Observation Architecture Report Trigger 2
OAREPORTTRIG6 - Observation Architecture Report Trigger 6
CEC0-0 - Customizable Event Creation 0-0
CEC1-0 - Customizable Event Creation 1-0
CEC1-1 - Customizable Event Creation 1-1
CEC2-0 - Customizable Event Creation 2-0
CEC2-1 - Customizable Event Creation 2-1
CEC3-0 - Customizable Event Creation 3-0
CEC3-1 - Customizable Event Creation 3-1
CEC4-0 - Customizable Event Creation 4-0
CEC5-0 - Customizable Event Creation 5-0
CEC5-1 - Customizable Event Creation 5-1

Register
CEC6-0 - Customizable Event Creation 6-0
CEC6-1 - Customizable Event Creation 6-1
CEC7-0 - Customizable Event Creation 7-0
CEC7-1 - Customizable Event Creation 7-1

The following Performance Statistics registers are power context save/restored:

Register
OAPERF_A0 - Aggregate Perf Counter A0
OAPERF_A0_UPPER - Aggregate Perf Counter A0 Upper DWord
OAPERF_A1 - Aggregate Perf Counter A1
OAPERF_A1_UPPER - Aggregate Perf Counter A1 Upper DWord
OAPERF_A2 - Aggregate Perf Counter 2
OAPERF_A2_UPPER - Aggregate Perf Counter A2 Upper DWord
OAPERF_A3 - Aggregate Perf Counter A3
OAPERF_A3_UPPER - Aggregate Perf Counter A3 Upper DWord
OAPERF_A4 - Aggregate Perf Counter A4
OAPERF_A4_UPPER - Aggregate Perf Counter A4 Upper DWord
OAPERF_A4_LOWER_FREE - Aggregate Perf Counter A4 Lower DWord Free
OAPERF_A4_UPPER_FREE - Aggregate Perf Counter A4 Upper DWord Free
OAPERF_A5 - Aggregate Perf Counter A5
OAPERF_A5_UPPER - Aggregate Perf Counter A5 Upper DWord
OAPERF_A6 - Aggregate Perf Counter A6
OAPERF_A6_UPPER - Aggregate Perf Counter A6 Upper DWord
OAPERF_A6_LOWER_FREE - Aggregate Perf Counter A6 Lower DWord Free
OAPERF_A6_UPPER_FREE - Aggregate Perf Counter A6 Upper DWord Free
OAPERF_A7 - Aggregate Perf Counter A7
OAPERF_A7_- Upper Aggregate Perf Counter A7 Upper DWord
OAPERF_A8 - Aggregate Perf Counter A8
OAPERF_A8_UPPER - Aggregate Perf Counter A8 Upper DWord
OAPERF_A9 - Aggregate Perf Counter A9
OAPERF_A9_UPPER - Aggregate Perf Counter A9 Upper DWord
OAPERF_A10 - Aggregate Perf Counter A10
OAPERF_A10_UPPER - Aggregate Perf Counter A10 Upper DWord
OAPERF_A11 - Aggregate Perf Counter A11
OAPERF_A11_UPPER - Aggregate Perf Counter A11 Upper DWord
OAPERF_A12 - Aggregate Perf Counter A12
OAPERF_A12_UPPER - Aggregate Perf Counter A12 Upper DWord
OAPERF_A13 - Aggregate Perf Counter A13
OAPERF_A13_UPPER - Aggregate Perf Counter A13 Upper DWord
OAPERF_A14 - Aggregate Perf Counter A14

Register
OAPERF_A14_UPPER - Aggregate Perf Counter A14 Upper DWord
OAPERF_A15 - Aggregate Perf Counter A15
OAPERF_A15_UPPER - Aggregate Perf Counter A15 Upper DWord
OAPERF_A16 - Aggregate Perf Counter A16
OAPERF_A16_UPPER - Aggregate Perf Counter A16 Upper DWord
OAPERF_A17 - Aggregate Perf Counter A17
OAPERF_A17_UPPER - Aggregate Perf Counter A17 Upper DWord
OAPERF_A18 - Aggregate Perf Counter A18
OAPERF_A18_UPPER - Aggregate Perf Counter A18 Upper DWord
OAPERF_A19 - Aggregate Perf Counter A19
OAPERF_A19_UPPER - Aggregate Perf Counter A19 Upper DWord
OAPERF_A19_LOWER_FREE - Aggregate Perf Counter A19 Lower DWord Free
OAPERF_A19_UPPER_FREE - Aggregate Perf Counter A19 Upper DWord Free
OAPERF_A20 - Aggregate Perf Counter A20
OAPERF_A20_UPPER - Aggregate Perf Counter A20 Upper DWord
OAPERF_A20_UPPER_FREE - Aggregate Perf Counter A20 Upper DWord Free
OAPERF_A20_LOWER_FREE - Aggregate Perf Counter A20 Lower DWord Free
OAPERF_A21 - Aggregate Perf Counter A21
OAPERF_A21_UPPER - Aggregate Perf Counter A21 Upper DWord
OAPERF_A22 - Aggregate Perf Counter A22
OAPERF_A22_UPPER - Aggregate Perf Counter A22 Upper DWord
OAPERF_A23 - Aggregate Perf Counter A23
OAPERF_A23_UPPER - Aggregate Perf Counter A23 Upper DWord
OAPERF_A24 - Aggregate Perf Counter A24
OAPERF_A24_UPPER - Aggregate Perf Counter A24 Upper DWord
OAPERF_A25 - Aggregate Perf Counter A25
OAPERF_A25_UPPER - Aggregate Perf Counter A25 Upper DWord
OAPERF_A26 - Aggregate Perf Counter A26
OAPERF_A26_UPPER - Aggregate Perf Counter A26 Upper DWord
OAPERF_A27 - Aggregate Perf Counter A27
OAPERF_A27_UPPER - Aggregate Perf Counter A27 Upper DWord
OAPERF_A28 - Aggregate Perf Counter A28
OAPERF_A28_UPPER - Aggregate Perf Counter A28 Upper DWord
OAPERF_A29 - Aggregate Perf Counter A29
OAPERF_A29_UPPER - Aggregate Perf Counter A29 Upper DWord
OAPERF_A30 - Aggregate Perf Counter A30
OAPERF_A30_UPPER - Aggregate Perf Counter A30 Upper DWord
OAPERF_A31 - Aggregate_Perf_Counter_A31
OAPERF_A31_UPPER - Aggregate Perf Counter A31 Upper DWord
OAPERF_A32 - Aggregate_Perf_Counter_A32

Register
OAPERF_A33 - Aggregate_Perf_Counter_A33
OAPERF_A34 - Aggregate_Perf_Counter_A34
OAPERF_A35 - Aggregate_Perf_Counter_A35
OAPERF_B0 - Boolean_Counter_B0
OAPERF_B1 - Boolean_Counter_B1
OAPERF_B2 - Boolean_Counter_B2
OAPERF_B3 - Boolean_Counter_B3
OAPERF_B4 - Boolean_Counter_B4
OAPERF_B5 - Boolean_Counter_B5
OAPERF_B6 - Boolean_Counter_B6
OAPERF_B7 - Boolean_Counter_B7
EU_PERF_CNT_CTL0 - Flexible EU Event Control 0
EU_PERF_CNT_CTL1 - Flexible EU Event Control 1
EU_PERF_CNT_CTL2 - Flexible EU Event Control 2
EU_PERF_CNT_CTL3 - Flexible EU Event Control 3
EU_PERF_CNT_CTL4 - Flexible EU Event Control 4
EU_PERF_CNT_CTL5 - Flexible EU Event Control 5
EU_PERF_CNT_CTL6 - Flexible EU Event Control 6
GPU_TICKS - GPU_Ticks_Counter

OA Interrupt Control Registers

The Interrupt Control Registers listed below all share the same bit definition. The bit definition is as follows:

Bit	Description
31:29	**Reserved. MBZ:** These bits may be assigned to interrupts on future products/steppings.
28	**Performance Monitoring Buffer Half-Full Interrupt:** For internal trigger (timer based) reporting, if the report buffer crosses the half full limit, this interrupt is generated.
27:0	**Reserved: MBZ (These bits must be never set by OA, these bit could be allocated to some other unit)**

- GT Interrupt 3 Definition
- IMR
- Bit Definition for Interrupt Control Registers

Performance Counter Report Formats

Counters layout for various values of select from the register:

Counters layout for various values of the "Counter Select" from the register:

Counter Select = 000

A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)	A-Cntr 11 (low dword)

Counter Select = 010

A-Cntr 10 (low dword)	A-Cntr 9 (low dword)	A-Cntr 8 (low dword)	A-Cntr 7 (low dword)	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
A-Cntr 18 (low dword)	A-Cntr 17 (low dword)	A-Cntr 16 (low dword)	A-Cntr 15 (low dword)	A-Cntr 14 (low dword)	A-Cntr 13 (low dword)	A-Cntr 12 (low dword)	A-Cntr 11 (low dword)
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-cntr 0
C-Cntr 7	C-Cntr 6	C-Cntr 5	C-Cntr 4	C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0

Counter Select = 101

Performance Counter Report Format 101b

Counter Select = 111

C-Cntr 3	C-Cntr 2	C-Cntr 1	C-Cntr 0	GPU_TICKS	CTX ID	TIME_STAMP	RPT_ID
B-Cntr 7	B-Cntr 6	B-Cntr 5	B-Cntr 4	B-Cntr 3	B-Cntr 2	B-Cntr 1	B-Cntr 0

Description of RPT_ID and other important fields of the layout:

Field	Description
GPU TICKS[31:0]	GPU_TICKS is simply a free-running count of render clocks elapsed used for normalizing other counters (e.g. EU active time), it is expected that the rate that this value advances will vary with frequency and freeze (but not lose its value) when all GT clocks are gated, GT is in RC6, and so on. **[Register] GPU_Ticks_Counter**

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Context ID[31:0]</td><td>This field carries the Context ID of the active context in render engine.
[31:0]: Context ID in Execlist mode of scheduling.
[31:12]: Context ID in Ring Buffer mode of scheduling, [11:0] must be ignored.</td></tr>
<tr><td>TIME_STAMP[31:0]</td><td>This field provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. This field has the same format at TIMESTAMP register defined in Vol1C.4 Render Command Streamer BSpec.</td></tr>
<tr><td>RPT_ID[31:0]</td><td>This field has several sub fields as defined below:
<table>
<tr><td>31:26</td><td>Reserved MBZ</td></tr>
<tr><td>25</td><td>Render Context Valid: When set indicates render context is valid and the ID is of the render context is set in “Context ID” field of report format.</td></tr>
<tr><td>24:19</td><td>Report Reason[5:0]:
Report_reason[0]: When set indicates current report is due to “Timer Triggered”.
Report_reason[1]: When set indicates current report is due to “Internal report trigger 1”.
Report_reason[2]: When set indicates current report is due to “Internal report trigger 2”.
Report_reason[3]: When set indicates current report is due to “Render context switch”.
Report_reason[4]: When set indicates current report is due to “GO transition from ‘1’ to ‘0’ ”.
Report_reason[5]: Reserved</td></tr>
<tr><td>18</td><td>Start Trigger Event:This bit is multiplexed from “Start Trigger Event-1” or “Start Trigger Event-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default Start Trigger Event-1 is outputted.</td></tr>
<tr><td>17</td><td>Threshold Enable: This bit is multiplexed from “Report Trigger Threshold Enable-1” or “Report Trigger Threshold Enable-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default “Report Trigger Threshold Enable-1” is outputted.</td></tr>
<tr><td>16</td><td>Timer Enabled</td></tr>
<tr><td>15:0</td><td></td></tr>
</table></td></tr>
</table>

Field	Description
RPT_ID[31:0]	Subfields of RPT_ID detailed below:

31:25	**squashed_slice_clock_frequency[6:0]**: Ratio encoding in this field can be decoded using the ratio encoding table.
24:19	**Report Reason[5:0]**: Report_reason[0]: When set indicates current report is due to “Timer Triggered”. Report_reason[1]: When set indicates current report is due to “Internal report trigger 1”. Report_reason[2]: When set indicates current report is due to “Internal report trigger 2”. Report_reason[3]: When set indicates current report is due to “Render context switch”. Report_reason[4]: When set indicates current report is due to “GO transition from ‘1’ to ‘0’ ”. Report_reason[5]: When set indicates the current report is due to Clock Ratio change between squashed Slice Clock frequency to squashed Unslice clock frequency.
18	**Start Trigger Event:**This bit is multiplexed from “Start Trigger Event-1” or “Start Trigger Event-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default Start Trigger Event-1 is outputted.
17	**Threshold Enable:** This bit is multiplexed from “Report Trigger Threshold Enable-1” or “Report Trigger Threshold Enable-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default “Report Trigger Threshold Enable-1” is outputted.
16	**Render Context Valid:** When set indicates render context is valid and the ID is of the render context is set in “Context ID” field of report format.
15:0	**Additional Report Flags:** Disable OA reports due to clock ratio change” is 1, these bits comprise of following: [10:9]: squashed_slice_clock_frequency [8:7] [8:0]: squashed_unslice_clock_frequency [8:0] Ratio encoding in this field can be decoded using the ratio encoding table.

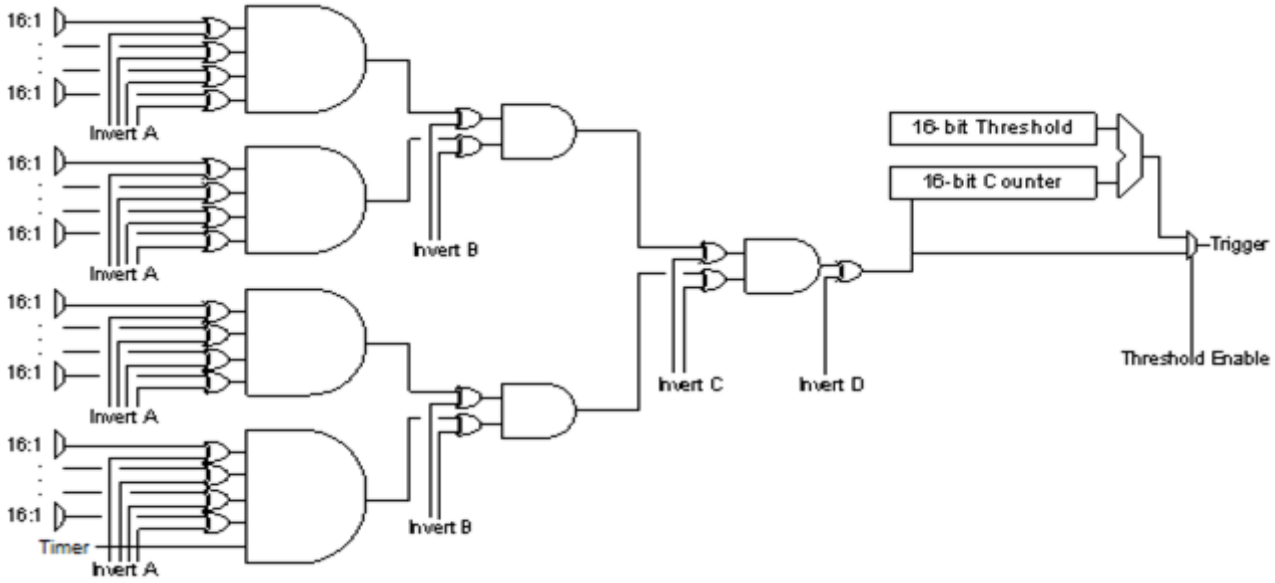
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>GPU TICKS[31:0]</td><td>GPU_TICKS is simply a free-running count of render clocks elapsed used for normalizing other counters (e.g. EU active time), it is expected that the rate that this value advances will vary with frequency and freeze (but not lose its value) when all GT clocks are gated, GT is in RC6, and so on.</td></tr>
<tr><td>TIME_STAMP[31:0]</td><td>This field provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. This field has the same format at TIMESTAMP register defined in Vol1C.4 Render Command Streamer BSpec.</td></tr>
<tr><td>RPT_ID[31:0]</td><td>This field has several sub fields as defined below:
<table>
<tr><td>31:26</td><td>Reserved MBZ</td></tr>
<tr><td>25</td><td>Render Context Valid: When set indicates render context is valid and the ID is of the render context is set in “Context ID” field of report format.</td></tr>
<tr><td>24:19</td><td>Report Reason[5:0]:
Report_reason[0]: When set indicates current report is due to “Timer Triggered”.
Report_reason[1]: When set indicates current report is due to “Internal report trigger 1”.
Report_reason[2]: When set indicates current report is due to “Internal report trigger 2”.
Report_reason[3]: When set indicates current report is due to “Render context switch”.
Report_reason[4]: When set indicates current report is due to “GO transition from ‘1’ to ‘0’ “.
Report_reason[5]: Reserved</td></tr>
<tr><td>18</td><td>Start Trigger Event:This bit is multiplexed from “Start Trigger Event-1” or “Start Trigger Event-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default Start Trigger Event-1 is outputted.</td></tr>
<tr><td>17</td><td>Threshold Enable: This bit is multiplexed from “Report Trigger Threshold Enable-1” or “Report Trigger Threshold Enable-2” based on the “Internal Report Trigger-1” or “Internal Report Trigger-2” asserted in the Report Reason respectively. “Internal Report Trigger-1” is given priority over “Internal Report Trigger-2”. By default “Report Trigger Threshold Enable-1” is outputted.</td></tr>
<tr><td>16</td><td>Timer Enabled</td></tr>
<tr><td>15:0</td><td></td></tr>
</table></td></tr>
<tr><td>RPT_ID[31:0]</td><td>Subfields of RPT_ID detailed below:
<table>
<tr><td>31:25</td><td>squashed_slice_clock_frequency[6:0]:
Ratio encoding in this field can be decoded using the ratio encoding table.</td></tr>
</table></td></tr>
</table>

Field	Description	
	24:19	**Report Reason[5:0]**: Report_reason[0]: When set indicates current report is due to "Timer Triggered". Report_reason[1]: When set indicates current report is due to "Internal report trigger 1". Report_reason[2]: When set indicates current report is due to "Internal report trigger 2". Report_reason[3]: When set indicates current report is due to "Render context switch". Report_reason[4]: When set indicates current report is due to "GO transition from '1' to '0' ". Report_reason[5]: : When set indicates the current report is due to Clock Ratio change between squashed Slice Clock frequency to squashed Unslice clock frequency.
	18	**Start Trigger Event:**This bit is multiplexed from "Start Trigger Event-1" or "Start Trigger Event-2" based on the "Internal Report Trigger-1" or "Internal Report Trigger-2" asserted in the Report Reason respectively. "Internal Report Trigger-1" is given priority over "Internal Report Trigger-2". By default Start Trigger Event-1 is outputted.
	17	**Threshold Enable:** This bit is multiplexed from "Report Trigger Threshold Enable-1" or "Report Trigger Threshold Enable-2" based on the "Internal Report Trigger-1" or "Internal Report Trigger-2" asserted in the Report Reason respectively. "Internal Report Trigger-1" is given priority over "Internal Report Trigger-2". By default "Report Trigger Threshold Enable-1" is outputted.
	16	**Render Context Valid:** When set indicates render context is valid and the ID is of the render context is set in "Context ID" field of report format.
	15:0	**Additional Report Flags:** When Disable OA reports due to clock ratio change" is 1, these bits comprise of following: [10:9]: squashed_slice_clock_frequency [8:7] [8:0]: squashed_unslice_clock_frequency [8:0] Disable OA reports due to clock ratio change**"** is 0**,** Ratio encoding in this field can be decoded using the ratio encoding table.

Performance Counter Reporting

When either the MI_REPORT_PERF_COUNT command is received or the internal report trigger logic fires, a snapshot of the performance counter values is written to memory. The format used by HW for such reports is selected using the Counter Select field within the **OACONTROL** register. The organization and number of report formats vary per project and are detailed in the following section. In the following layouts, the RPT_ID is always stored in the lowest addressed DWORD.

OA contains logic to control when performance counter values are reported to memory. This functionality is controlled using the OA report trigger and OA start trigger registers. More detailed register descriptions are included in the Hardware Programming interface. The block diagram below illustrates the logic these registers control.



Note that counters which are 40 bits wide in BDW are split in the report format into low DWORD and high byte chunks for simplicity of HW implementation as well as SW-friendly alignment of report data. The performance counter read logically done before writing out report data for these 40-bit counters is guaranteed to be an atomic operation, the counter data is simply swizzled as it is being packed into the report.

Due to OA being located in render power well, render power well shutdown must be disabled for proper performance counter functionality when only workloads which do not exercise render logic (e.g. certain media workloads) are running.

Aggregating Counters

The table below described the desired high-level functionality from each of the aggregating counters.

Note that there is no counter of 2x2s sent to pixel shader, this is based on the assumption that the pixel shader invocation pipeline statistics counter increments for partially lit 2x2s as well and hence does not require a duplicate performance counter.

Counter #	Event	Description
A1	# of Vertex Shader Threads Dispatched	Count of VS threads dispatched to EUs **[Register] Aggregate Perf Counter A1**
A2	# of Hull Shader Threads Dispatched	Count of HS threads dispatched to EUs **[Register] Aggregate Perf Counter A2**
A3	# of Domain Shader Threads Dispatched	Count of DS threads dispatched to EUs **[Register] Aggregate Perf Counter A3**
A4	# of GPGPU Threads Dispatched	Count of GPGPU threads dispatched to EUs **[Register] Aggregate Perf Counter A4**
A5	# of Geometry Shader Threads Dispatched	Count of GS threads dispatched to EUs **[Register] Aggregate Perf Counter A5**
A6	# of Pixel Shader Threads Dispatched	Count of PS threads dispatched to EUs **[Register] Aggregate Perf Counter A6**
A7	Aggregating EU counter 0	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A7**
A8	Aggregating EU counter 1	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A8**
A9	Aggregating EU counter 2	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A9**
A10	Aggregating EU counter 3	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A10**
A11	Aggregating EU counter 4	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A11**
A12	Aggregating EU counter 5	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A12**

Counter #	Event	Description
A13	Aggregating EU counter 6	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A13**
A14	Aggregating EU counter 7	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A14**
A15	Aggregating EU counter 8	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A15**
A16	Aggregating EU counter 9	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A16**
A17	Aggregating EU counter 10	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A17**
A18	Aggregating EU counter 11	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A18**
A19	Aggregating EU counter 12	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A19**
A20	Aggregating EU counter 13	User-defined (details in Flexible EU Event Counters section) **[Register] Aggregate Perf Counter A20**
A21	2x2s Rasterized	Count of the number of samples of 2x2 pixel blocks generated from the input geometry before any pixel-level tests have been applied. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A21**
A22	2x2s Failing Fast pre-PS Tests	Count of the number of samples failing fast "early" (i.e. before pixel shader execution) tests (counted at 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A22**
A24	2x2s Killed in PS	Number of samples entirely killed in the pixel shader as a result of explicit instructions in the kernel (counted in 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A24**

Counter #	Event	Description
A25	2x2s Failing post-PS Tests	Number of samples that entirely fail "late" tests (i.e. tests that can only be performed after pixel shader execution). Counted at 2x2 granularity. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A25**
A26	2x2s Written To Render Target	Number of samples that are written to render target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A26**
A27	Blended 2x2s Written to Render Target	Number of samples of blendable that are written to render target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) **[Register] Aggregate Perf Counter A27**
A28	2x2s Requested from Sampler	Aggregated total 2x2 texel blocks requested from all EUs to all instances of sampler logic. **[Register] Aggregate Perf Counter A28**
A29	Sampler L1 Misses	Aggregated misses from all sampler L1 caches. Please note that the number of L1 accesses varies with requested filtering mode and in other implementation specific ways. Hence it is not possible in general to draw a direct relationship between A28 and A29. However, a high number of sampler L1 misses relative to texel 2x2s requested frequently degrades sampler performance. **[Register] Aggregate Perf Counter A29**
A30	SLM Reads	Total read requests from an EU to SLM (including reads generated by atomic operations). **[Register] Aggregate Perf Counter A30**
A31	SLM Writes	Total write requests from an EU to SLM (including writes generated by atomic operations). **[Register] Aggregate Perf Counter A31**
A32	Other Shader Memory Accesses	Aggregated total requests from all EUs to memory surfaces other than render target or texture surfaces (e.g. shader constants). **[Register] Aggregate Perf Counter A32**

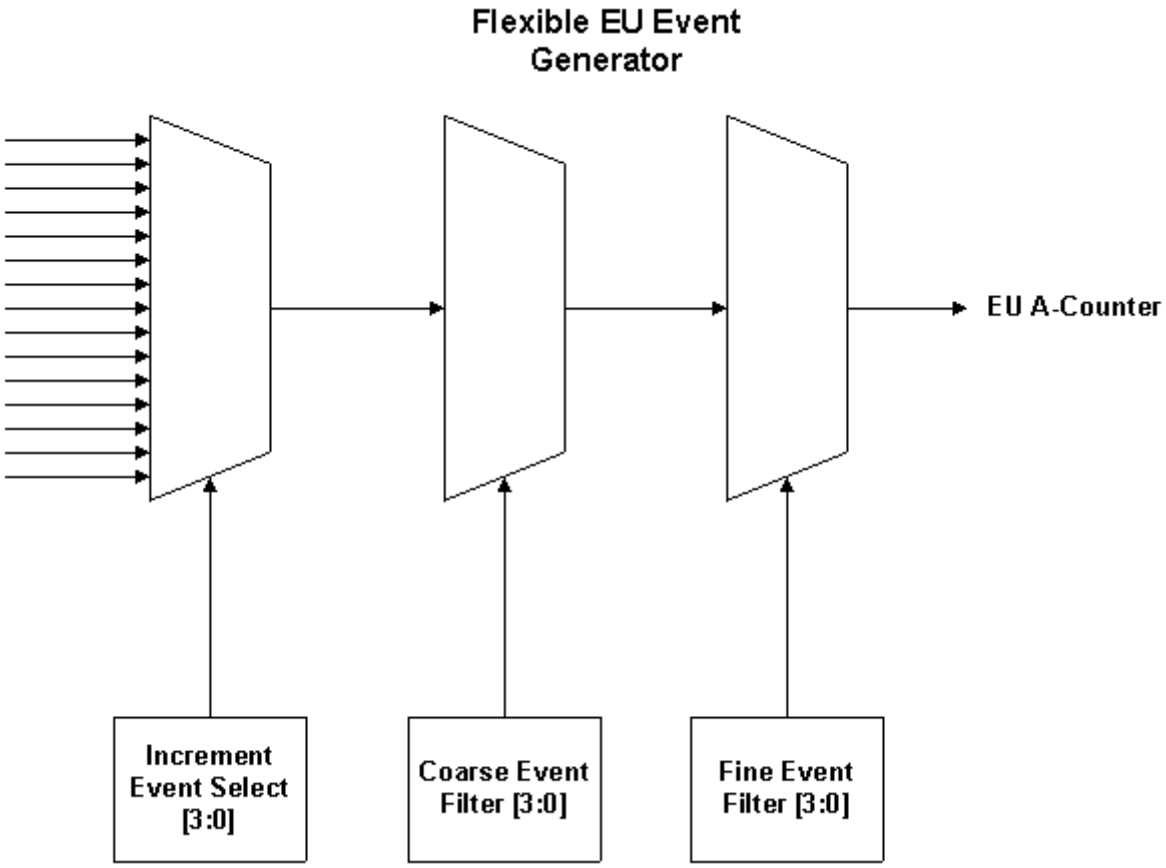
Counter #	Event	Description
A33	Other Shader Memory Accesses That Miss First-Level Cache	Aggregated total requests from all EUs to memory surfaces other than render target or texture surfaces (e.g. shader constants) that miss first-level cache. **[Register] Aggregate Perf Counter A33**
A34	Atomic Accesses	Aggregated total atomic accesses from all EUs. This counter increments on atomic accesses to both SLM and URB. **[Register] Aggregate Perf Counter A34**
A35	Barrier Messages	Aggregated total kernel barrier messages from all Eus (one per thread in barrier). **[Register] Aggregate Perf Counter A35**

Flexible EU Event Counters

Since EU performance events are most interesting in many cases when aggregated across all EUs and many interesting EU performance events are limited to certain APIs (e.g. hull shader kernel stats only applicable when running a DX11+ workload), BDW adds some additional flexibility to the aggregated counters coming from the EU array.

The following block diagram shows the high-level flow that generates each flexible EU event.

Note that no support is provided for differences between flexible EU event programming between EUs because the resulting output from each EU is eventually merged into a single OA counter anyway.



Supported Increment Events

Increment Event	Encoding	Notes
EU FPU0 Pipeline Active	0b0000	Signal that is high on every EU clock where the EU FPU0 pipeline is actively executing a Gen ISA instruction.
EU FPU1 Pipeline Active	0b0001	Signal that is high on every EU clock where the EU FPU1 pipeline is actively executing a Gen ISA instruction.
EU SEND Pipeline Active	0b0010	Signal that is high on every EU clock where the EU send pipeline is actively executing a Gen ISA instruction. Only fine event filters 0b0000,0b0101, 0b0110, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
EU FPU0 & FPU1 Pipelines Concurrently Active	0b0011	Signal that is high on every EU clock where the EU FPU0 and FPU1 pipelines are both actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Some EU Pipeline Active	0b0100	Signal that is high on every EU clock where at least one EU pipeline is actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000,0b0101, 0b0110, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
At Least 1 Thread	0b0101	Signal that is high on every EU clock where at least one thread is loaded but no

Increment Event	Encoding	Notes
Loaded But No EU Pipeline Active		EU pipeline is actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Threads loaded integrator == max threads for current HW SKU	0b1000	Implies an accumulator which increases every EU clock by the number of loaded threads, signal pulses high for one clock when the accumulator exceeds a multiple of the number of thread slots (e.g. for a 8-thread EU, signal pulses high every clock where the increment causes a 3-bit accumulator to overflow). Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event. **Note:** There were no C steppings for the BDW GPU, so this increment event is supported for BDW:*:E0 and later. **Note:** A thread slot is considered to be "loaded" once the EU receives the transparent header. Hence, time spent transferring additional thread header data phases (e.g. push constants and vertex attribute data) will count towards thead occupancy.

Supported Coarse Event Filters

Coarse Event Filter	Encoding	Notes
No mask	0b0000	Never masks increment event.
VS Thread Filter	0b0001	For increment events 0b0000/0b0001/0b0010, masks increment events unless the FFID which dispatched the currently executing thread equals FFID of VS.
HS Thread Filter	0b0010	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of HS.
DS Thread Filter	0b0011	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of DS.
GS Thread Filter	0b0100	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of GS.
PS Thread Filter	0b0101	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of PS.
TS Thread Filter	0b0110	For increment events 0b0000/0b0001/0b0010, masks increment event unless the FFID which dispatched the currently executing thread equals FFID of TS.

Coarse Event Filter	Encoding	Notes
Row = 0	0b0111	Masks increment event unless the row ID for this EU is 0 (control register is in TDL so only have to check within quarter-slice).
Row = 1	0b1000	Masks increment event unless the row ID for this EU is 1 (control register is in TDL so only have to check within quarter-slice).

Fine Event Filters

Fine Event Filter	Encoding	Notes
None	0b0000	Never mask increment event.
Cycles where hybrid instructions are being executed	0b0001	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are hybrid instructions.
Cycles where ternary instructions are being executed	0b0010	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are ternary instructions.
Cycles where binary instructions are being executed	0b0011	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are binary instructions.
Cycles where mov instructions are being executed	0b0100	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are mov instructions.
Cycles where sends start being executed	0b0101	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are send start of dispatch. Note that if this fine event filter is used in combination with increment events not related to the EU send pipeline (e.g. FPU0 active), the associated flexible event counter will increment in an implementation-specific manner.
EU# = 0b00	0b0111	Masks increment event unless the EU number for this EU is 0b00.
EU# = 0b01	0b1000	Masks increment event unless the EU number for this EU is 0b01.
EU# = 0b10	0b1001	Masks increment event unless the EU number for this EU is 0b10.
EU# = 0b11	0b1010	Masks increment event unless the EU number for this EU is 0b11.

Flexible EU Event Config Registers

EU_PERF_CNT_CTL0 - Flexible EU Event Control 0

EU_PERF_CNT_CTL1 - Flexible EU Event Control 1

EU_PERF_CNT_CTL2 - Flexible EU Event Control 2

EU_PERF_CNT_CTL3 - Flexible EU Event Control 3

EU_PERF_CNT_CTL4 - Flexible EU Event Control 4

EU_PERF_CNT_CTL5 - Flexible EU Event Control 5

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 15: Scaler Format Converter (SFC)

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

SFC Overview **1**

Ultra Low-Power Usages 1

SFC Commands Definition **2**

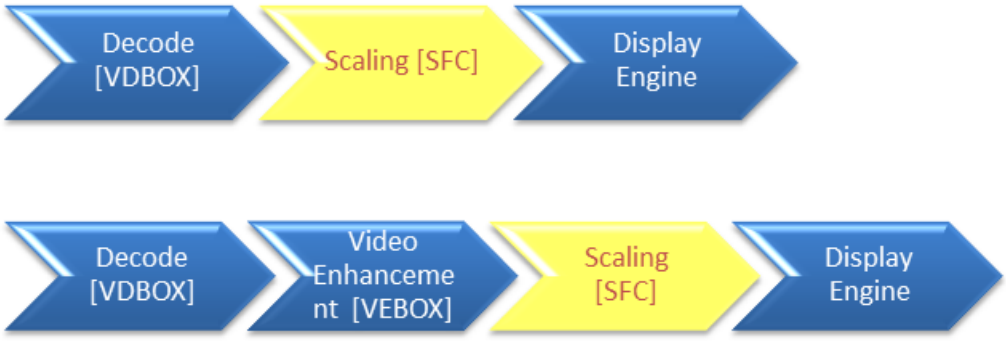
SFC Overview

Scaler & Format Converter (SFC) pipeline is introduced as a multi-format scaling engine to accelerate several media usages and achieve ultra low-power video playback.

Ultra Low-Power Usages

Several power saving techniques are brought into consideration and incorporated into the architecture of SFC pipeline: offloading from EU to fixed function to reduce Cdyn power, memory traffic reduction to lower IO and DDR power, and native surface support between acceleration engines.

Two ultra low–power playback modes are introduced to achieve sub 1-Watt solution: Decode->Scale-> Display (sprite) and Decode-> Image Enhancement-> Scale-> Display (sprite).



- In these two usages, SFC is fed by the decoder (VDBOX) and image enhancer (VEBOX) directly instead of writing to memory and read back from memory. A direct data bus is added between VD-to-SFC and VE-to-SFC. SFC will also include an internal store buffer to capture overlap pixel data between column/rows. In another word, the only IO traffic to DDR is the final scaled surface writes. All input and intermediate traffics related to SFC engine are confined inside GT and not expose to external components.
- EU-less usage: SFC is a fixed function engine architects to run concurrently along VDBOX or VEBOX. i.e. Decode and scaling will be happening at the same time, or Image enhancement and scaling will be occurring at the same time. It saves power by offloading the scaling workload off the media render engine to this dedicated engine which is much smaller.
- In both cases, scaling operation is the last processing step before final pixels are presented by the display engine. SFC is designed to generate output format native to display engine. This reduces the memory traffic caused by elimination of the extra memory copy used to convert the format incompatibility between engines. In addition, SFC supports 90 degree clockwise rotation of the final pixel surface for tablet space.

- SFC pipeline is chained together with VDBOX and VEBOX with direct interface and ability to run concurrently. VDBOX/VEBOX sends control parameters and pixel data directly to SFC through direct interface. This help reduces the IO and package power by eliminating the traffic to memory, and allows VD/VE to run concurrently along with SFC pipeline. SFC pipeline is a shared resource that can be called and accessed by VDBOX or VEBOX. A lock must be placed and granted with an acknowledgement ahead of transferring data to SFC. On completion, the lock must be removed to free up the shared resource (SFC).

SFC Commands Definition

This section contains definitions for commands used with the scaler and format converter (SFC). These commands are sent from the VDBOX/VEBOX to the SFC pipeline.

SFC_AVS_LUMA_Coeff_Table

SFC_AVS_CHROMA_Coeff_Table

SFC_AVS_STATE

SFC_FRAME_START

SFC_IEF_STATE

SFC_LOCK

SFC_STATE

Intel® Open Source HD Graphics and Intel Iris™ Plus Graphics

Programmer's Reference Manual

For the 2016 - 2017 Intel Core™ Processors, Celeron™ Processors, and Pentium™ Processors based on the "Kaby Lake" Platform

Volume 16: Workarounds

January 2017, Revision 1.0

Creative Commons License

You are free to Share - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

Table of Contents

Workarounds **1**

Display Workarounds **19**

Workarounds

This page lists all BSpec narrative workarounds for Gen9 (SKL/BXT/GLK/GLV). Note that the functional area for each item is listed below, so you can search on this value or other content on this page. Also note that a "BSpec ID" has been assigned to make it easier to reference these items.

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0538	KMD		WaSkipStolenMemoryFirstPage	WA to skip the first page of stolen memory due to sporadic HW write on *CS Idle	KBL: SIWA_UNTIL_B0
0550	KMD		WaForceEnableNonCoherent	Must Force Non-Coherent whenever executing a 3D context. This is a workaround for a possible hang in the unlikely event a TLB invalidation occurs during a PSD flush. Set masked bit 4 in 0x7300 during boot.	KBL: UNTIL_D0
0551	KMD		WaDisableMidThreadPreempt	Disable GPGPU thread-level (a.k.a. mid-thread) preemption on parts (until B0) since validation was minimal on those parts.	KBL: SIWA_FOREVER
0556	KMD		Wa4x4STCOptimizationDisable	HIZ/STC hang in hawx frames. W/A: Disable 4x4 RCPFE-STC optimization and therefore only send one valid 4x4 to STC on 4x4 interface. This will require setting bit 6 of reg. 0x7004. Must be done at boot and all save/restore paths.	SIWA_FOREVER
0566	KMD		WaModifyVFEStateAfterGPGPUPreemption	GPGPU preemption hang or corruption issue. This WA must be applied before re-submitting a GPGPU preempted workload. SW needs to do two things.	KBL: SIWA_UNTIL_D0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				1. SW should detect if VFE unit in the context image have LRI commands to 0x54AC and 0x54B0. SW does that by checking if the LRI header at context image offste (0x0E90) <in VFE> is 0x11001003 and the two MMIO offsets in this LRI is 0x54AC and 0x54B0. If so, do this- D0 (LRI header): 0x1100_1003 -> 0x1100_ 1001 D1 (1st MMIO offset): 0x0000_54AC -> Leave this dword untouched D2 (1st MMIO offset value): 54AC_DATA -> Leave this dword untouched D3 (2nd MMIO offset): 0x0000_54B0 -> Overwrite to 0x0 D4 (2nd MMIO offset value): 54B0_DATA -> Overwrite to 0x0 2. Apart from it SW must also increment the Batch Buffer Address by Walker Command Length so that CS moves to the command following it. Size of Walker should be 0xF *	

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				sizeof(DWORD) = 0x3C Bytes	
0572	KMD	RTL	WaFlushCoherentL3CacheLinesAtContextSwitch	Coherent L3 cache lines are not getting flushed during context switch which is causing issues like corruption. Need to set bit 21 of MMIO b118, then send PC with DC flush and then reset bit 21 of b118. This programming sequence needs to be part of the indirect context WA BB	SIWA_FOREVER
0574	KMD		WaDisableSDEUnitClockGating	WA for GPGPU workload hang for which requirement is to disable SDE Unit clock gating. This is done by setting bit 14 of MMIO 9430.	KBL: SIWA_UNTIL_B0
0581	KMD		WaSetVfGuardbandPreemptionVertexCount	Workaround for potential 3d preemption bug that can cause data corruption. Driver should write to register 0x83A4 (preemption vertex count) and set a value of 0x20. At boot, write 0xffff0020 to 0x83a4 (it's a masked register).	KBL: SIWA_UNTIL_B0
0590	KMD		WaSkipInvalidSubmitsFromOS	For Invalid submits from OS - simply report fence completion without submitting the DMA buffer to GPU.	SIWA_FOREVER
0592	3D		SKL_Z_16X_ALIGNMENT	TileY cannot be used for MSAA 8x or 16x with multiple MIP levels. The fallback is to use TileYF. Tile YS would also be suitable.	
0673	3D		WaStallBeforePostSyncOpOnGPGPU	Preemption mid-thread focused test failures.	SIWA_FOREVER

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				WA: PIPECONTROL command with "Command Streamer Stall Enable"must be programmed prior to programming MI_SEMAPHORE_SIGNAL command with Post-Sync Operation set in GPGPU mode of operation (i.e when PIPELINE_SELECT command is set to GPGPU mode of operation). PIPECONTROL command with "Command Streamer Stall Enable"must be programmed prior to programming MI_ATOMIC command with Post-Sync Operation set in GPGPU mode of operation (i.e when PIPELINE_SELECT command is set to GPGPU mode of operation). PIPECONTROL command with "Command Streamer Stall Enable"must be programmed prior to programming a PIPECONTROL command with Post Sync Op in GPGPU mode of operation (i.e when PIPELINE_SELECT command is set to GPGPU mode of operation).	

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0675	3D		WaFlushBefore3DSTATEGS	GS_SIMD8_OTHANDLE_RELAX test hanging due to an issue in gs_trg clock gating logic. WA: Add state_osb_statedv into trg_cg equation.	SIWA_FOREVER
0677	3D		WaDisableLosslessCompressionForSampleL	Sampler Throughput drop with lossless enabled for 0% & 50%, compression tests with 100%bypass. WA: Disabe double-fetch.	SIWA_FOREVER
0680	3D		WaDisableSamplerL2BypassForTextureCompressedFormats	RTL DM producing Xs for SC output. WA: Disable Bypass on some of the compressed formats.	SIWA_FOREVER
0681	3D		WaCompressedResourceSamplerPbeMediaNewHashMode	BXT GFXDRV: Hot spotting issue with render target compression. WA: Align lossless compressed resource allocations to 2MB or have fixed virtual addresses. This WA is specifically for steppings where HW will not fix.	KBL:SIWA_FOREVER
0683	3D		WaCompressedResourceDisplayNewHashMode	BXT GFXDRV: Hot spotting issue with render target compression. WA: Align lossless compressed resource allocations to 2MB or have fixed virtual addresses. This WA is specifically for steppings where HW will not fix.	KBL:SIWA_FOREVER

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0689	3D		WaPipeControlBeforeVFCacheInvalidationEnable	Vf randomly decodes nullprim packets as state packets causing illegal internal states in it. WA: B2B control packets to be sent when VS_cache_enable is programmed to be enabled. First control packet with bit11 as '0' and the next control packet with bit11 as '1'.	SIWA_FOREVER
0693	3D		WaEnableTiledResourceTranslationTables	Support for Sparse Tiled Resources under SVM	KBL:SIWA_FOREVER
0695	3D		WaIndependentAlphaBlend	GFX SV: optHizClear test miscompares in B0 silicon. WA: Fixed optimization case for independent alpha to look at src values for RGB and Alpha values separately. Fixed the src0dest1 case for alpha component to look at src_alpha_is_zero instead of src_is_zero.	KBL:SIWA_FOREVER
0696	3D		WaBindlessSurfaceStateModifyEnable	Missing "Size Modify Enable" Bit For Bindless Sizes in STATE_BASE_ADDRESS. WA: The suggested WA is that when NOT setting the modify enable bit for Bindless Surface State Base Address, program the dword length to “Eh”instead of “11h”and zero out the last 3 DW or not send them.	SIWA_FOREVER

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0699	3D		WaAtomicFlushOnInterfaceDescriptor	GPGPU Preemption - Hang on Barrier WA: VFE need to make sure that all previous indirect prefetches are done after VFE state command. MI_ATOMIC_FLUSH need to be programmed between ID load commands or curbe load commands if it back to back.	KBL:SIWA_FOREVER
0701	3D		WaStructuredBufferAsRawBufferOverride	BSpec restricts Data Port access using Untyped Surface Read/Write to RAW surface format only. WA: Raw format support added in svsmunit.	KBL:SIWA_FOREVER
0702	3D			False Inexact IEEE flag set by SP to UQ mov. WA: Cannot use flags with float2int convertion with non-RTZ rounding mode.	
0714	Media	VTQ		Trellis quantization performance improvements introduced a bug. The bug results in Hang condition when Chroma Trellis is disabled and Trellis is enabled. The bug is fixed in KBL B0.	KBL A0
0744	3D		WaDisableRTReadsfor1DSurfaces	Failing to implement the following WA will lead to pixel corruptions due to accesses to WA : If RenderSurfaceState::Surface Type == SURFTYPE_1D, SW must ensure that Pixel Shaders do not generate RT reads.	KBL:SIWA_A0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0752	3D		WaSamplerResponseLengthMustBeGreaterThan1	Dcs_pwc_rc signal is not set when notify clear comes out of phase WA: disable MMIO reads from GW & all sampler sends in GPGPU workloads with response length of 1.	KBL:SIWA_FROM_A0
0756	Media	HEVC		VP9 Decoder 8Kx8K SV test failing with miscompare at 2nd frame. WA: Frame size supported in BXT is 4Kx4K.	KBL:SIWA_FROM:A0
0827				Moved to Display Workaround page	
0828				Moved to Display Workaround page	
0831	KMD		WaDisableSamplerPowerBypassForSOPingPong	SI can get stuck ping ponging rows in a 2-2-2 fashion instead of 1-1-1 leading to a ~3% performance reduction. WA: Disable sampler power bypass to avoid negatively impacting SO 'ping-pong' performance.	KBL: SIWA_FOREVER
0837	GAM		WaSpuriousIOMMUFaults	GT GAM HW prefetches context (or extended context) entry when a context is loaded, root pointer is updated or when there is a context cache flush. This prefetch happens without a memory transaction from the context. On this prefetch, if the context entry is a NULL (P=0), HW will generate a fault – invalid context entry. However, it is legal to have a NULL context entry, as long as no	KBL:ALL

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				memory references are done via that context entry. WA: To avoid these spurious DMA faults, SW should mark the context entry as present (not a NULL entry), and mark the page tables as not present.	
0838	GTI			MGSR hang when MsgCh cycle arrives a couple clocks after IOSF SB shadow request. This is being brought up into the SW WA section for completeness. 0xD00[3:1] already indicate that bits should be set by SW. WA: Enable 0x0D00[3:1] fixes in SW for all Gen9 products.	KBL:ALL
0851				Moved to Display Workaround page	
0852				Moved to Display Workaround page	
0856				Moved to Display Workaround page	
0857				Moved to Display Workaround page	
0858	3D	Sampler		CSS AUX surfaces must not be on pages that can fault.	KBL:SIWA_UNTIL_D1
0859				Moved to Display Workaround page	
0860	GTI	GAM		When only the coherent flush FSM and HDC invalidation FSMs are active, GAM can turn off the clocks after receiving COH flush done even if HDC	KBL:A0,B0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				FSM is active. This can cause invalidation end to not be sent to HDC after HDC FSM completes. WA: Turn off GAM clock gating by setting 0x9400[22] to '1	
0864	GTI	GAM		When only the coherent flush FSM and HDC invalidation FSMs are active, GAM can turn off the clocks after receiving COH flush done even if HDC FSM is active. This can cause invalidation end to not be sent to HDC after HDC FSM completes. WA: Turn off GAM clock gating by setting 0x9400[22] to '1	KBL:A0/B0
0865	GTI	GAM		The dynamic credit sharing between engines has an issue which can cause the credits to be lost. When sufficient credits are lost the BW through GAM can get severely limited and can eventually block all the transactions and hang. WA: Turn off dynamic credit sharing by setting 0x4AB8[28] to '1	KBL:A0/B0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0866	GTI	GAM		Issue here is with QBI and GFXIDLE - If QBI is in progress and we get GFXIDLE, it will be ACKED and it will make CTX inactive. Once CTX will go inactive it will make all pasid/ctxid match flags zero which is one of the condition for generating hdc_end for QBI. QBI will end but HD will not see end due to this Flag. WA: CS to send invalidation to GAM before indicating IDLE, GAM can handle future QBI's without involving HDC and this bug wont be exposed. Insert Pipecontrol with invalidation flag set at the end of the ring, batch. Same is required while pre-empting the existing WL.	KBL:A0/B0
0867	GTI	GAM		When HDC/EUs are set to Faul&Halt mode, the commit interface between HDC and EUs are disabled. If coherency is enabled, we can have coherent cycles in the fabric when an external invalidation arrives, and can lead into the GAM coherent deadlock scenario if GAM resources are full. GAM deadlock coherent_flush will	KBL:ALL

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				kick in to resolve the deadlock. However, this requires EUs to re-dispatch the coherent cycles that were killed. If HDC/EU are in Fault&Halt mode, this re-start is not possible from the EUs. Do not use Fault&Halt as the fault mode in HDC/EU	
0868				Moved to Display Workaround page	
0870				Moved to Display Workaround page	
0871				Moved to Display Workaround page	
0872	3D	CS		Global Workaround Batch Buffer will not execute when enabled and Execution List mode is enabled. WA: Do not enable Global WABB when in Execution List mode.	KBL:ALL (except GT4)
0873				Moved to Display Workaround page	
0875	GTI	GAM		When there is a TLB invalidation with a Fence, GAM blocks the arbiters to drain the fabric for the Fence. Once the fabric is drained, the TLBs are invalidated, as well as a HDC TLB invalidation request is sent to the HDC. GAM expects an ack for this from HDC. If there are cycles in HDC that has already used the address translations, they need to be drained from HDC	KBL:A0/B0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				before it can send the Ack. GAM is supposed to unblock the arbiters at this stage to allow for these cycles to drain. However, when TRTT is enabled, the arbiter is GATR does not get unblocked until the HDC inval Ack is completed. This causes a deadlock where HDC cannot send the Ack as there are cycles that needs to drain. In general when CS issues a Fence Inval, there should not be any pending cycles in HDC if a CS stall is put in before and after the fence inval. However, in the case of lite restore, Fence Inval can happen asynchronously, potentially leaving pending cycles. The WAs for this issue are: 1. Disable TRTT – so the fabric does not continue to get blocked in GATR after the fence 2. Disable lite restore – so we do not have asynchronous fence inval events. Also need to have a CS stall before and after CS fence inval. SW is planning on having both Was available, and #1 as the default.	

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0876	GTI	CS	WaForceCsStallOnTimestampQueryOrDepthCount	Due to known HW issue specific to GT4; on a specific incident (few HW events must happens on the same clock under certain programming conditions) address, data and control fields corresponding to a PIPECONTROL command will get registered wrongly in hardware. Following this incident Fence Reports, Depth Statistic Report (Occlusion Query) and time_stamp reports will get corrupted leading to OS/KMD/UMD hangs. **Workaround (option-1):** PIEPCONTROL command programmed with "Post Sync Operation" set to "Write Timestamp" or "Write Depth Query" must also set "Command Streamer stall Enable" to '1'. **Workaround (option-2):** Software must memorize the event of programming "Post Sync Operation" set to "Write Timestamp" or "Write Depth Query" and must set "Pipecontrol Flush Enable" on next PIPECONTROL programmed. Note: Since the passing on the memorized event between UMD and	KBL:GT4 (all GT4 SKU's)

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				KMD (ring buffer and batch buffer) is difficult, one way it could be addressed in following way. KMD must always program the first PIPECONTROL being programmed in the ringbuffer following the MI_BATCH_BUFFER_START with "Pipecontrol Flush Enable" set. KMD must always program PIPECONTROL with "Pipecontrol Flush Enable" set prior to programming MI_BATCH_BUFFER_START in the ring buffer. **OR** UMD must always program the first PIPECONTROL in the batch buffer with "Pipecontrol Flush Enable" set and must always program the last command in every dispatch of the batch buffer to a PIPECONTROL with "Pipecontrol Flush Enable" set.	
0877	3D	Pixel Shader	WaSendDummyConstantsForPS	Hang possible when pixel shader dispatched with only header phases (R0-R2) WA: Enable a non-header phase (e.g. push constant) when dispatch would have been header-only.	KBL:ALL
0880				Moved to Display Workaround page	
0881				Moved to Display Workaround page	

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0883				Moved to Display Workaround page	
0884				Moved to Display Workaround page	
0889				Moved to Display Workaround page	
0890				Moved to Display Workaround page	
0893				Moved to Display Workaround page	
0898	Media			WA: In encoder mode, create tiles either in horizontal or vertical direction but not in both directions.	KBL:ALL
0908	SFC	SFC Crop Limitation for VEBOX+SFC Mode	WaDisableSFCSrcCrop	Below are the cases to switch from SFC to Render for VEBOX+SFC mode Case 1. ((SurfaceHeight > 1120) && (Top > 1120)) Case 2. ((SurfaceHeight > 1120) && (Bottom < SurfaceHeight)) Case 3. ((SurfaceHeight > 1120) && (Left > 0)) Case 4. ((SurfaceHeight > 1120) && (Right < SurfaceWidth))	KBL A0
0909	GS PrimID bug with Tessellation	Peter Doyle		GS Clock gating must be disabled under the following conditions: Tessellation enabled, GS enabled, GS PrimitiveID enabled.	KBL: ALL

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0910	GAM	Niran Cooray		When DINVE=0 in the ext context entry PASID state table is not used. But, HW reads the PASIDSTPTR from the context entry and goes through SL translations, if NESTE is enabled. If the SL translation returns invalid, all transactions from that engine is considered invalid. WA: Have a valid GPA for the PASIDSTPTR which has a valid GPA->HPA mapping in the SL tables. This SL translation need to have R=W=1 permissions	KBL:UNTIL_GT2: A0-C0 KBL:UNTIL_GT3: D0 KBL:UNTIL_GT4: E0
0915	HDC	Atomic Counter		Hang occurs with Atomic Counter message with binary operations (i.e. have a separate data operand), in some dynamic load situations. WA: Replace use of hidden counter with an explicit counter location, and then use a typed or untyped Dword Atomic operation message instead.	KBL:ALL
0917	GTI	GAM		Reset of an engine (non render) does not complete when there is a continuous stream of traffic comes from another engine. Another flavor of this issue is with VEBox, where VEBox reset does not complete when there's continuous polling happening	KBL: UNTIL_F0

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				from VECS. The second flavor is seen with MI_SEMAPHORE_WAIT. WA: 1. for the first case (reset of A blocked due to traffic from B), the only workaround is to stall/stop engine B that is continuously generating traffic. 2. for the second case, the polling period need to be increased to let HW identify the fabric idleness and complete the Go=0 Ack for reset. Increase the poll interval in VECS_SEMA_WAIT_POLL (0x1A24C). Setting this to ius resolves the issue.	

Display Workarounds

This page lists all BSpec workarounds for Display. Note that the functional area for each item is listed below, so you can search on this value or other content on this page. Also note that a "BSpec ID" has been assigned to make it easier to reference these items.

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0342	Display	DisplayPort		Not a workaround, this is required programming for all projects. DP MST requires certain VC payload size values. VC payload must be multiple of 4 in x1 lane config, 2 in x2, 1 in x4. See transcoder M/N Values.	All
0346	Display	DisplayPort		Aux channel transactions get intermittent NAK errors with some receivers. WA: Do not use the 400us Aux timeout. Increase DDI_AUX_CTL bits 27:26 Time out timer value to at least 600us 01b when doing DDI aux transactions. Merged with #0347 This becomes standard required programming for all subsequent projects.	All
0371	Display	Panel fitter	WaPanelFitterDownscale	Not a workaround, this is required programming for all projects. When using panel fitter downscaling (pipe source size is larger than panel fitter window size) the maximum supported pixel rate will be reduced by the downscale amount, and watermarks must be adjusted. Use panel fitter scale amount when calculating maximum pixel rate and watermarks.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0373	Display	Panel power sequencing	WaVDDOverrideT4Power	Not a workaround, this is required programming for all projects. When software clears the panel power sequencing VDD override bit from 1 to 0 (disable VDD override) it must ensure that T4 power cycle delay is met before setting the bit to 1 again. Use software timers to ensure T4 delay is met or use full panel power enable and not the VDD override.	All
386	Display	PSR		PSR single frame update - When single frame update is enabled, the PSR CRC must be disabled for panel compatibility.	All
0387	Display	PSR		PSR single frame update - Mask register write events when using single frame update. This becomes standard required programming for all subsequent projects.	All
0388	Display	PSR		PSR power saving - Mask PSR max timeout when PSR CRC is enabled. This becomes standard required programming for all subsequent projects.	All
0471	Power	FBC		First line of FBC getting corrupted when FBC compressed frame buffer offset is programmed to zero. Command streamers are doing flush writes to base of stolen. WA: New restriction to program FBC compressed frame buffer offset to at least 4KB. See	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				FBC_CFB_BASE in North Display Registers. Additionally software must not use this first 4KB of graphics data stolen memory for anything else. This becomes standard required programming for all subsequent projects.	
0477	Power	IPC		IPC (Isoch Priority Control) may cause underflows. WA: Do not enable IPC in register ARB_CTL2	All
0482	Power	PSR		The following sequence is needed when disabling PSR Single Frame Update to workaround cases where the remote frame buffer update indicator becomes stuck if there is a single frame update exit event (pipe register write) around the time that single frame update is being disabled. 1. Set 0x420B0 bit 11 to mask off flips. 2. Set PIPE_MISC bit 21 to mask off pipe register writes. 3. Clear SRD_CTL bit 31 to 0 to disable PSR. 4. Wait for 2 vertical blanks for PSR to completely disable. 5. Clear SRD_CTL bit 30 to 0 disable Single Frame Update. 6. Restore PIPE_MISC bit 21.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				7. Restore 0x420B0 bit 11. PSR can now be re-enabled if needed.	
0484	Power	Watermarks		Watermark memory latency data retrieved from GT driver mailbox data may not account for memory read latency. Depending on mailbox response, add 2 microseconds to the result for each level to compensate. This becomes standard required programming for all subsequent projects.	All
0485	Power	Watermarks		Notable Bspec correction. Not a workaround. Watermark memory latency data retrieved from GT driver mailbox data had incorrect dword ordering in earlier versions of the BSpec. BSpec is now corrected to show the first set of data has latency levels 0-3 and second set has levels 4-7. This becomes standard required programming for all subsequent projects.	All
0529	Display	FBC		Corruption in some cases when FBC is enabled and the plane surface format is in linear, tile Y legacy, or tile Yf WA: Display register 4208Ch bit 13 must be set to 1b and bits 12:0 must be programmed with the compressed buffer stride value. The compressed buffer stride must be calculated using the following	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				equation: Compressed buffer stride = ceiling [(at least plane width in pixels) / (32 * compression limit factor)] * 8 At least plane width = a value greater than or equal to the width of the plane. Software may choose to use a greater value in order to handle cases where the plane width is changing from frame to frame, especially because 4208C is not double-buffered and can't be changed on the fly while FBC is enabled. Compression limit factor is either 1, 2 or 4 based on the FBC_CTL Compression Limit field.	
0531	Display	Render Compression		Render decompression is broken when plane widths greater than 3840 are used with horizontal panning. WA: When the render compression is enabled with plane width greater than 3840 and horizontal panning (Start X Position in the PLANE_OFFSET register is not 0), the stride programmed in the PLANE_STRIDE register must be multiple of 4.	All
0562	Power	FBC		FBC sometimes causes screen corruption with package C states. WA: 'FBC Watermark Disable' bit in ARB_CTL register must be set to 1b.	All
0827	Display	Planes		Switching the plane format from NV12 to RGB and leaving system idle results in display underrun and corruption.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				WA: Set the bit 15 & bit 19 to 1b in the CLKGATE_DIS_PSL register for the pipe in which NV12 plane is enabled.	
0828	Display	PSR2		PSR2 screen corruptions when multiple regions are updated in a single frame. WA: Set 0x42080 bit 3 = 1 before enabling PSR2. The register can safely remain set when PSR2 is disabled.	All
0840	Display	Watermarks SAGV		Programming needed for SAGV to prevent underflows in multi-display scenarios. See Display Watermark Programming - Watermark Calculations section. This becomes standard required programming for subsequent core projects.	All
0851	Display	FBC	WaFbcNukeOn3DBlt	To prevent missed invalidations around the time FBC is being enabled, FBC render tracking must use the "Render Tracking With Nuke" method. See Frame Buffer Compression page.	All
0856	Display	Memory Bandwidth		Display underflow with high resolutions and multiple displays. WA: Restrict display configurations to fit within system memory bandwidth threshold specifed in the Display Mode Set -> Display Resolution Support page. Increase watermarks at some system memory bandwidth thresholds. See Display Watermark Calculations.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				The restriction to fit within memory bandwidth threshold is required programming for all subsequent projects. The threshold can change from project to project and will be listed on the Display Resolution Support page.	
0857	Display	Planes		Display underrun issues with Y & Yf Tiling. WA: Set the bit 13 of MMIO register 0x46430 to 1b.	All
0859	Display	FBC	WaFbcWakeMemOn	This workaround helps to achieve better idle power savings when FBC is enabled. WA: Set ARB_CTL FBC Memory Wake to 1'b1.	All
0868	Display	GMBUS		4 block EDID failures. WA: Set 0x4653C[14] = 0x1 to disable clock gating when the north display GMBUS function is used. Set 0xC2020[31]=0x1 to disable clock gating when the south display GMBUS function is used.	All
0873	Display	FBC		Screen corruption observed with FBC when the front buffer is updated by host modify. WA: To prevent missed host invalidations around the time FBC is being enabled, enable Nuke on modify. Set bit 23 of MMIO register 0x43224 to 1'b1.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
0880	Display	Clocks		Timeout for display cdclk mailbox programming increased from 1ms to 3ms to account for some corner cases that can exceed 1ms. This becomes standard required programming for all subsequent projects.	ALL
0881	Display	Clocks		Display PLL workaround for signal integrity when using the 2.16 or 4.32 GHz eDP link rate together with spread spectrum clocking. The GT driver pcode mailbox has to be programmed to override some DPLL internal values. See Display -> North Display Engine Registers -> Clocks -> Port Clock Programming.	All
0883	Display	FBC	WaFbcHighMemBwCorruptionAvoidance	When FBC is enabled sometimes screen corruptions/system hangs observed under high memory bandwidth conditions. WA: Set the bit 8 of MMIO register 0x43224 to 1b. Set ARB_CTL FBC Memory Wake to 1b (from #0859). This memory wake setting is also preferred for better power savings with FBC.	KBL:*:A KBL:*:B
0893	Display	Memory Bandwidth		Display underflow with high resolutions and multiple displays when using dual channel single rank memory. WA: Increase watermarks at some system memory bandwidth thresholds. See Display Watermark Calculations.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
1106	Display	NV12, Rotation, Horizontal flip		Display corruption/color shift observed when using NV12 with 270 rotation or 90 rotation + horizontal flip. WA: NV12 with 270 rotation or 90 rotation + horizontal flip requires the programmed plane height to be a multiple of 4.	All
1107	Display	Transition Watermark		Transition watermarks must not be enabled.	All
1110	Display	FBC + PSR/PSR2		Missing flips when FBC is enabled with PSR link off/PSR2 deep sleep scenarios. WA: When FBC is enabled with PSR/PSR2, set bit 30 of MMIO register 0x420CC to 1b.	KBL:*:A KBL:*:B
1125	Display	Watermarks, render compression		Render compression watermark adjustment. This was a late addition to the Bspec, but not previously tracked as a workaround. See the Watermark Calculations page for details. WA: If Render Decompression enabled and latency level 0: Result Blocks = Result Blocks + Y tile minimum. Then ensure that the result blocks for higher latency levels are all at least as high as the new level 0.	All
1126	Display	Watermarks		Watermark adjustment. This was a late addition to the Bspec, but not previously tracked as a workaround. See the Watermark Calculations page for details.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				WA: If latency level 1 through 7 and Y tile: Result Blocks = Result Blocks + Y tile minimum; Result Lines = Result Lines + Minimum Scanlines for Y tile If latency level 1 through 7 and not Y tile: Result Blocks = Result Blocks + 1	
1131	Display	VTd		Display underflow and flicker when VTd is enabled with a frame buffer having a mix of VTd 4K, 2MB and 1GB pages. WA: When VTd is enabled, all the extra dummy PTE entries must be mapped to pages of same size (4K, 1GB or 2MB). All the extra entries shall map to the same dummy page. This dummy page padding is required immediately after the display active fetch area. If any of the display features (like panning, clipping, collage displays, planar YUV surfaces) can't meet the above requirement it must not be enabled. See PLANE_SURF restrictions for Extra Page Table Entries allocation requirements.	All
1132	Display	64bpp		Bandwidth limitations in the display data buffer output can cause underflow with 64bpp pixel formats when multiple planes are enabled at high resolutions. The 64 bits per pixel source pixel format can only be used on a single plane and when only a single pipe is enabled.	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
1134	Display	GMBUS		GMBUS bit bashing failures because GPIO pin is unexpectedly in busy state. WA: When starting bit bashing, first write GPIO_CTL = 0x00000404.	All
1136	Display	PSR		Display underrun with PSR single frame update and planes with less than watermark level 7. WA: When using PSR single frame update, all enabled planes must have enabled up to watermark level 7. If any plane cannot support level 7, then single frame update cannot be enabled.	KBL:*:A KBL:*:B
1141	Power	IPC		IPC (Isoch Priority Control) may cause underflows. WA: When IPC is enabled, watermark latency values must be increased by 4us across all levels. This brings level 0 up to 6us..	All
1142	Display			Underruns found with FBC, but also could impact cases without FBC. WA: Before enabling display planes or cursor, program 0x42084 bits 14:13=10b and 0x42080 bit 22=1b. Keep this value while planes or cursor are enabled. The value is ignored while planes and cursor are all disabled. The same programming is used with and without FBC.	KBL:SIWA_FROM_C0
1143	HDMI			For HDMI the voltage swing programming for a port is locked at the value used when the port is first enabled after boot or package C10. Override programming is needed to allow the swing to change after that point, such as for DP++ ports or	All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings
				electrical validation. 1. Just before enabling DDI_BUF_CTL for HDMI in the port enable sequence. 2. Set override for this DDI. o DDIB: 0x420C0 19:18 = 11b o DDIC: 0x420C4 19:18 = 11b o DDID: 0x420C8 19:18 = 11b o DDIA: 0x420CC 19:18 = 11b 3. Wait 1us for DDI to pick up the swing value. 4. Clear override for this DDI to 00b. 5. Continue with enabling DDI_BUF_CTL.	

1144	Audio			DisplayPort audio corruption or video underflow with some clock frequencies and audio configurations.
				
				WA:
				
				If (DP port width x4 AND link rate HBR2 AND CDCLK frequency < 432 MHz) {
				
				If (audio clock frequency >96 KHz OR 8 audio channels used OR audio configuration unknown) {
				
				DisplayPort audio cannot be enabled at this CDCLK frequency.
					All

BSpec ID	Functional Area/Component		Workaround Name	Workaround Description	Valid Steppings

				}
				
				Else {
				
				Set register (0x420C0 for audio on transcoder A, 0x420C4 for transcoder B, 0x420C8 for transcoder C) bit 13 to 1 before enabling DisplayPort audio.
				
				}
				
				}	